AccuTerm 7 GUI

Table of Contents

1 AccuTerm 7 GUI Development

1.1 Introduction

AccuTerm 7 GUI (Graphical User Interface) consists of four components:

- AccuTerm GUI Library (GUIBP)
- Transport mechanism (AccuTerm)
- GUI runtime
- GUI designer

A MultiValue GUI application is written in MultiValue BASIC as a stand-alone program, or as a set of called subroutines (see Multiple Application Model). You can also create a dialog box subroutine which can be called from any program, either GUI or green-screen. The AccuTerm GUI Library provides all the necessary routines to create and manipulate GUI forms (using the GUI designer is not necessary). The GUI application you create for the AccuTerm GUI environment is a true Windows event driven application, with the look and feel of traditional Widows applications.

The GUI designer is provided to aid in the layout of graphical elements and choose fonts, colors, styles, etc. The GUI designer creates a "template" which saves the designed elements and their properties as an item in a host data file. A single subroutine, ATGUIRUNMACRO, is used to realize the template at runtime.

The GUI designer also includes and integrated code generator to aid in the creation of GUI applications. The code generator creates MultiValue BASIC code customized for the GUI application being designed. The code generator creates all of the required code for initialization, event processing, event decoding and leaves "stubs" for your own custom event handling.

The simplest way to create a GUI application is to use the GUI designer and code generator, then simply add your custom event handling code.

1.2 The GUI Library

The AccuTerm GUI library provides a collection of MultiValue BASIC subroutines which interact with the GUI runtime. Each of the routines is described below. An INCLUDE item is provided which defines constants and EQUATEs for use with the library.

This library is normally loaded into the GUIBP host file, and is installed during installation of the AccuTerm host file transfer programs. If you need to install this at a later time, type RUN FTBP LOAD-ACCUTERM-GUIBP at TCL.

The structure of a typical AccuTerm GUI program is:

```
Perform application initialization
Initialize the GUI environment (ATGUIINIT2)
Load GUI project for application (ATGUIRUNMACRO)
Show the first (and maybe other) form (ATGUISHOW)
Loop
    Wait for user-interface event (ATGUIWAITEVENT)
    Decode event
    Call event handler
Until Quit event Repeat
Shutdown GUI environment (ATGUIDELETE, ATGUISHUTDOWN)
Stop
```

The typical event handler will query the GUI application for the values of controls, validate data, retrieve or update data from the host data base, update values of controls, show or hide forms, activate specific controls or forms, etc.

When assigning a numeric value to size or position properties, you can use real numbers that contain a decimal point. The decimal point character must be the period (dot) character. When assigning the Value or DefaultValue property, use the natural, locale-specific character for a decimal point (not necessarily a dot, except in USA).

The following is a list of each of the GUI library routines, organized into functional groups:

GUI Environment Functions
ATGUIINIT2 - initializes the GUI environment
ATGUISHUTDOWN - closes the GUI environment
ATGUISUSPEND - suspends processing the GUI application
ATGUIRESUME - resumes processing the GUI application after it was suspended

GUI Object Creation Functions
ATGUICREATEAPP - creates a new application
ATGUICREATEFORM - creates a new form
ATGUICREATEEDIT - creates an edit (text-box) control
ATGUICREATELABEL - creates a label control
ATGUICREATELIST - creates a list-box control
ATGUICREATECOMBO - creates a combo-box control
ATGUICREATEOPTION - creates an option-group (radio buttons) control
ATGUICREATECHECK - creates a check-box control
ATGUICREATEBUTTON - creates a command button control
ATGUICREATEGRID - creates a grid control (great for associated multi-values!)
ATGUICREATEPICTURE - creates a picture control

ATGUICREATEFRAME2 - creates a frame control
ATGUICREATETABGRP - creates a tab-group control
ATGUICREATETAB - creates a tab control (must be within a tab-group)
ATGUICREATETREE - creates a tree control
ATGUICREATEGAUGE - creates a gauge, or progress-bar control
ATGUICREATEMENU - creates a main (form) or popup (context) menu
ATGUICREATETOOLBAR - creates a toolbar
ATGUICREATESTATUSBAR - creates a status bar
ATGUICREATETIMER - creates a timer
ATGUIDELETE - deletes a control, form or application
Global **Objects** - Root object, Screen object and Printer object

GUI State Functions
ATGUIACTIVATE - makes a form or control the active GUI object
ATGUISHOW - makes an application, form, control or item visible
ATGUIHIDE - makes an application, form, control or item invisible
ATGUIENABLE - enables a control or item
ATGUIDISABLE - disables a control or item
ATGUIMOVE - moves a form or control to a new position
ATGUIGETACTIVE - returns the ID of the active control

GUI Property Functions
ATGUISETPROP - sets the value of a property of an application, form or control
ATGUIGETPROP - returns the value of a property of an application, form or control
ATGUISETPROPS - sets the value of multiple properties of one or more GUI objects on a form
ATGUIGETPROPS - returns the values of multiple properties of one or more GUI objects on a form
ATGUISETITEMPROP - sets the value of a property of a specific item in a GUI object
ATGUIGETITEMPROP - returns the value of a property of a specific item in a GUI object
ATGUILOADVALUES - loads the values of a list of controls
ATGUIGETVALUES - returns the values of a list of controls
ATGUIGETUPDATES - returns a list of updated controls and their updated values
ATGUIRESET - resets a form (and all its controls) or control to its default value(s)
ATGUICLEAR - clears the data value of a control
ATGUIINSERT - inserts an empty row into a grid, list or combo-box control
ATGUIINSERTITEMS - inserts one or more items into a grid, list or combo-box at a specific row
ATGUIREMOVE - deletes a row from a grid, list or combo-box control
ATGUIREMOVEITEMS - deletes one or more items from a grid, list or combo-box control beginning at a specific row
Standard Properties - properties that are used by most GUI objects, like Left, Top, etc.

GUI Event Processing Functions
ATGUIWAITEVENT - wait for a user-interface event to occur
ATGUICHECKEVENT - wait a specified time for a user-interface event to occur
ATGUIPOSTEVENT - "posts" an event into the event queue (used in the Multiple Application Model)

GUI Macro Functions
ATGUIBEGINMACRO - begin recording a macro
ATGUIENDMACRO - ends macro recording
ATGUIRUNMACRO - plays a recorded macro or template created with the GUI designer

GUI Utility Functions
ATGUIPRINT2 - prints the current form on the default Windows printer
ATGUIHELP - displays help page in dedicated browser window

ATGUIMSGBOX - displays a warning or information message
ATGUIINPUTBOX - prompts for single-line text input
ATGUIFILEDIALOG - displays the standard "open" dialog and returns the selected file
ATGUICOLORDIALOG - displays the standard color dialog
ATGUIFONTDIALOG - displays the standard font dialog

1.2.1 GUI Environment Functions

1.2.1.1 ATGUIINIT2

The ATGUIINIT2 routine must be called before any other GUI functions are used. This routine checks that the correct version of AccuTerm is running and initializes the GUISTATE variable which must be passed all other GUI functions. The routine returns the version number of the GUI runtime engine.

Calling syntax:

```
CALL ATGUIINIT2(APPVER, SVRVER, GUIERRORS, GUISTATE)
```

Input arguments:

APPVER	version of the application. This is often taken from the project record<2,2> and is checked against the GUI Library (host programs) version and runtime engine (server) version. An error results if the application version is greater than either the library version or the runtime version. The current library and runtime version is 1.4.

Output arguments:

SVRVER	version of the runtime engine
GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

Note: the ATGUIINIT2 routine replaces the obsolete ATGUIINIT *routine.*

1.2.1.2 ATGUISHUTDOWN

The ATGUISHUTDOWN routine may be called to terminate the GUI server before the QUIT event has been received. This routine closes the GUI application and allows normal terminal operations to resume. This routine can be called at any time. After calling ATGUISHUTDOWN, you must call ATGUIINIT2 to start another GUI application.

Calling syntax:

```
CALL ATGUISHUTDOWN
```

Output arguments: none

1.2.1.3 ATGUISUSPEND

The ATGUISUSPEND routine suspends the currently running GUI applications and allows normal terminal function. ATGUIRESUME resumes the suspended GUI applications. The GUISTATE variable must be maintained between the ATGUISUSPEND and ATGUIRESUME calls. The GUI application is hidden while suspended.

Calling syntax:

```
CALL ATGUISUSPEND(GUIERRORS, GUISTATE)

CALL ATGUIRESUME(GUIERRORS, GUISTATE)
```

Input arguments:

GUISTATE	ATGUIRESUME only

Output arguments:

GUISTATE	ATGUISUSPEND only
GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

1.2.1.4 ATGUIRESUME

The ATGUISUSPEND routine suspends the currently running GUI applications and allows normal terminal function. ATGUIRESUME resumes the suspended GUI applications. The GUISTATE variable must be maintained between the ATGUISUSPEND and ATGUIRESUME calls. The GUI application is hidden while suspended.

Calling syntax:

```
CALL ATGUISUSPEND(GUIERRORS, GUISTATE)

CALL ATGUIRESUME(GUIERRORS, GUISTATE)
```

Input arguments:

GUISTATE	ATGUIRESUME only

Output arguments:

GUISTATE	ATGUISUSPEND only
GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

1.2.2 GUI Object Creation Functions

1.2.2.1 ATGUICREATEAPP

The ATGUICREATEAPP routine creates an application. The type of application, SDI (single document interface) or MDI (multiple document interface), must be specified. An application is the first GUI object that must be created. After creating an application, forms and controls can be created.

Calling syntax:

```
CALL ATGUICREATEAPP(APPID, EVENTMASK, TYPE, CAPTION, HELP, RTNMODE,
      SCALE,
      LEFT, TOP, WIDTH, HEIGHT, GUIERRORS, GUISTATE)
```

Input arguments:

APPID	application identifier (string)
EVENTMASK	should specify GEQUIT; if MDI app, also specify GECLOSE to catch main window close event
TYPE	type of application: 0 for SDI application, 1 for MDI application
CAPTION	application caption (displays in the MDI main window title bar)
HELP	if this argument ends with ".hlp" or ".chm", it specifies the Windows help file for the application and the help type is 0. If it is NULL, the help type is 1 and GEHELP events are handled by the host. Otherwise the help type is 2 and this argument specifies the URL of the help Contents or Index page.
RTNMODE	special handling of the **ENTER** key: 0 = normal 1 = same as **TAB** key 2 = same as **TAB** key except when active control is a Multi-line Edit control or a Command Button.
SCALE	coordinate scale for application: 0 = default (characters) 1 = twips 2 = points 3 = pixels 4 = characters 5 = inches 6 = millimeters 7 = centimeters
LEFT	horizontal position of the MDI main window relative to the screen (MDI app only)
TOP	vertical position of the MDI main window (MDI app only)
WIDTH	width of the MDI main window (MDI app only)
HEIGHT	height of the MDI main window (MDI app only)

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

Properties supported by this item:

GPBACKCOLOR	background color (default color is ApplicationWorkspace)
GPENABLED	non-zero if application is enabled
GPVISIBLE	non-zero if application is visible
GPPICTURE	filename or URL of image to be displayed as the MDI workspace background
GPCAPTION	application caption (displays in the MDI main window title bar)
GPICON	filename or URL of the application icon. This icon is used for the MDI main window icon, and becomes the default icon for all forms in the application. It is displayed in the **ALT+TAB** window when switching Windows applications.
GPSTYLE	MDI taskbar style: 0 = no MDI taskbar 1 = MDI taskbar buttons show child window title 2 = MDI taskbar buttons show child window state and title
GPHELPTYPE	type of help for the application: 0 = Windows help file (.hlp or .chm files) 1 = HTML document (host supplies HTML formatted text) 2 = HTML reference (topic ID is a URL)
GPHELPFILE	filename for Windows help file for application
GPHELPID	URL of applicaton Help Contents or Help Index page (help type = 2)
GPRTNEQTAB	special handling of the Enter key: 0 = normal 1 = same as Tab key 2 = same as Tab key except when active control is a Multi-line Edit control or a Command Button.
GPSCALE	coordinate scale for application: 0 = default (characters) 1 = twips 2 = points 3 = pixels 4 = characters 5 = inches 6 = millimeters 7 = centimeters
GPDESCRIPTION	application description (shown in the About box). Use subvalue marks to separate multiple lines.
GPCOPYRIGHT	application copyright notice (shown in the About box). Use subvalue marks to separate multiple lines.
GPAUTHOR	application author (shown in the About box)
GPVERSION	application version (shown in the About box)
GPLOGO	filename or URL of logo image displayed in the About box
GPTIMEOUT	time in seconds that host may process an event before the "Busy" status panel is displayed. Default is 2. Set this property to zero to prevent the busy panel from being shown.

GPMSGTEXT	Busy status panel message text. Default is "Processing... please wait".
GPAUTOSEL	non-zero if text fields are automatically selected when activated
GPNOAUTOTIPS	set to 1 to prevent tree controls from automatically showing truncated node text in a tooltip window; this property must be set before the tree control is created!
GPSTATUS	returns status information about the application: if COL = 0, returns the number of visible forms; if COL = 1, returns the ID of the currently active control; if COL = 2, returns a multi-valued list of child objects. The returned list can be restricted to return only Form objects by calling ATGUIGETPROP with ROW = 1, otherwise, if ROW = 0, the MDI application menu or toolbar will be included in the list.
GPWINDOWSTATE	0 = normal, 1 = minimized, 2 = maximized (MDI application only)

Events supported by this item (sum the desired events to form the EVENTMASK argument):

GEQUIT	the GUI application has terminated – end of event processing
GERESIZE	the MDI form has been resized, maximized or minimized: GUIARGS<1,1> = width GUIARGS<1,2> = height GUIARGS<1,3> = window state
GEACTIVATE	a form that belongs to the application has been activated: GUIARGS<1,1> = ID of the application being deactivated
GEDEACTIVATE	a different application has been activated: GUIARGS<1,1> = ID of the application being activated
GEHELP	the user has requested help. The host program should call the ATGUIHELP routine to display the requested help page: GUIARGS<1,1> = 0 if the user clicked the *Help menu item, 1 if the F1 key was pressed, 2 if a TOPIC hyperlink (in the currently displayed help page) was clicked, or 3 if post GUIARGS<1,2> = topic ID GUIARGS<1,3> = ID of the active control GUIARGS<1,4> = "post data" from the html page

1.2.2.2 ATGUICREATEFORM

The ATGUICREATEFORM routine creates a *form*. The type of form may be fixed, sizable or a dialog box. A *form* is a container for controls, which implement the user interface. Certain *form* property values (font, color) are used as the default values for controls created on the form. When a form is initially created, it is "hidden". To display the form, call ATGUISHOW.

Calling syntax:

```
CALL ATGUICREATEFORM(APPID, FORMID,  EVENTMASK, CAPTION, TYPE, LEFT,
      TOP,
      WIDTH, HEIGHT, GUIERRORS, GUISTATE)
```

Input arguments:

APPID	identifies which application the form belongs to

FORMID	form identifier
EVENTMASK	should specify GECLOSE
CAPTION	form caption
TYPE	type of form: 0 = fixed size form 1 = resizable form 2 = modal dialog box 4 = fixed size non-child form 5 = resizable non-child form
LEFT	horizontal position of the form relative to the screen (SDI) or the MDI main window (MDI), or “auto” to center the form
TOP	vertical position of the form relative to the screen (SDI) or the MDI main window (MDI), or “auto” to center the form
WIDTH	width of the form
HEIGHT	height of the form

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

GUISTATE

Properties supported by this item:

GPFORECOLOR	foreground (text) color (default color is WindowText)
GPBACKCOLOR	background color (default color is 3DFace)
GPFONTNAME	name of default font (default is Windows default)
GPFONTSIZE	font size in points (default is Windows default)
GPFONTBOLD	non-zero to use boldface font
GPFONTITALIC	non-zero to use italic font
GPENABLED	non-zero if form is enabled
GPVISIBLE	non-zero if form is visible
GPHELPID	help topic ID or URL
GPPICTURE	filename or URL of an image to be displayed as the form’s background
GPCAPTION	form caption
GPICON	filename or URL of form icon (default is application icon)
GPSTATUS	returns status information about the form: if COL = 2, returns a multi-valued list of child objects (controls which have the form as their parent).
GPWINDOWSTATE	0 = normal, 1 = minimized, 2 = maximized
GPDROPIDS	drop target IDs (see Drag and Drop)

Events supported by this item (sum the desired events to form the EVENTMASK argument):

GECLOSE	the Close button (or File Exit menu) was clicked.
GEACTIVATE	the form has become the active form. GUIARGS<1,1> is the ID of the form being deactivated.
GEDEACTIVATE	the form is no longer the active form. GUIARGS<1,1> is the ID of the form being activated.
GERESIZE	the form has been resized, maximized or minimized: GUIARGS<1,1> = width GUIARGS<1,2> = height GUIARGS<1,3> = window state Note: fixed size forms may receive this event if the user minimizes or restores the form to the taskbar.
GEDRAGDROP	the user has dropped an acceptable drag source on the form: GUIARGS<1,1> = ID of the drag source that was dropped on the form GUIARGS<1,2,1> = X coordinate within drag source where the drag began GUIARGS<1,2,2> = Y coordinate within drag source where the drag began GUIARGS<1,2,3... > = drag source details (depends on type of drag source object) GUIARGS<1,3,1> = X coordinate in form (drop target) where source was dropped GUIARGS<1,3,2> = Y coordinate in form (drop target) where source was dropped

1.2.2.3 ATGUICREATEEDIT

The ATGUICREATEEDIT routine creates an *edit* (or text box) *control*. The style of the edit control may be single line, multi line , password or single line with spin (up/down) buttons. An *edit control* is used to allow the user to enter and edit text. The multi-line form of the control allows the user to enter free-form text. Multi-line edit controls normally perform automatic word-wrap, and display a vertical scroll bar. A non-wrapping style is available which displays both vertical and horizontal scroll bars. The *value* of an *edit control* is the text string. For multi-line controls, lines are separated by value marks (only "hard" end-of-lines). The password style edit control displays an asterisk for each character the user types.

Calling syntax:

```
CALL ATGUICREATEEDIT(APPID, FORMID, CTRLID, PARENTID, EVENTMASK, STYLE,
       LEFT, TOP,
       WIDTH, HEIGHT, GUIERRORS, GUISTATE)
```

Input arguments:

APPID	identifies which application control belongs to
FORMID	identifies which form control belongs to
CTRLID	control identifier
PARENTID	identifies a “parent” container control, such as a frame, if any
EVENTMASK	list of events this control responds to
STYLE	edit control style: 0 = single line

	1 = multi-line with automatic word-wrapping 2 = password 3 = multi-line without automatic word-wrapping 4 = single line with spin (up/down) buttons
LEFT	horizontal position of the control relative to its parent form or frame
TOP	vertical position of the control relative to its parent form or frame
WIDTH	width of the control
HEIGHT	height of the control

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

GUISTATE

Properties supported by this item:

GPFORECOLOR	foreground (text) color (default is form's FORECOLOR)
GPBACKCOLOR	background color (default color is WindowBackground)
GPFONTNAME	name of default font (default is form's FONTNAME)
GPFONTSIZE	font size in points (default is form's FONTSIZE)
GPFONTBOLD	non-zero to use boldface font
GPFONTITALIC	non-zero to use italic font
GPFONTUNDERLINE	non-zero to use underlined font
GPENABLED	non-zero if control is enabled
GPVISIBLE	non-zero if control is visible
GPHELPID	help topic ID or URL
GPBORDER	0 = no border; 1 = flat border; 2 = normal border (default)
GPREADONLY	non-zero if control is read-only
GPDATATYPE	specify one of the enumerated data types
GPREQUIRED	non-zero to require a value for this control
GPALIGN	0 = left, 1 = right, 2 = center
GPMAXLEN	0 if no limit, otherwise specifies the maximum number of characters to accept
GPMAXLINES	0 if no limit, otherwise specifies the maximum number of lines to accept (multi-line only)
GPSELSTART	starting position of selected text
GPSELLENGTH	length of selected text (zero if no selection)
GPSELRANGE	selection starting position in propval<1,1>, length in propval<1,2>
GPSTYLE	edit control style: 0 = single line 1 = multi-line with automatic word-wrapping 2 = password 3 = multi-line without automatic word-wrapping 4 = single line with spin (up/down) buttons
GPDRAGID	drag source ID (not necessarily the control's CTRLID). When an edit

	control is dropped, GUIARGS<1,2,3> contains the character position in the edit control where the mouse button was pressed.
GPDOPIDS	drop target IDs (list of drag source IDs that are allowed to be dropped on this control)
GPDEFVAL	default value of control (also sets current value)
GPVALUE	value of control

Events supported by this item (sum the desired events to form the EVENTMASK argument):

GECHANGE	the user changed the control's value – this event is fired for each change (e.g. keystroke). GUIARGS<1> contains the modified value of the control.
GEVALIDATE	the user is attempting to move to another control after modifying the text in the control. Use this event to validate the control's new value. If validation fails, use ATGUIACTIVATE to re-activate this control. GUIARGS<1,1> = ID of the control triggering the event GUIARGS<2> = modified value of the control
GECLICK	the user clicked the left mouse button on this control.
GECONTEXT	the user clicked the right mouse button on this control.
GEDBLCLICK	the user double-clicked the left mouse button on this control.
GEACTIVATE	the control has become the active control. GUIARGS<1,1> is the ID of the control being deactivated.
GEBTNCLICK	the user clicked the up or down spin button: GUIARGS<1,1> = 0 if the "down" button was clicked or 1 if the "up" button was clicked GUIARGS<1,2> = modified value of the control
GEDEACTIVATE	the control is no longer the active control. GUIARGS<1,1> is the ID of the control being activated.
GEDRAGDROP	the user has dropped an acceptable drag source on the control: GUIARGS<1,1> = ID of the drag source that was dropped on the control GUIARGS<1,2,1> = X coordinate within drag source where the drag began GUIARGS<1,2,2> = Y coordinate within drag source where the drag began GUIARGS<1,2,3... > = drag source details (depends on type of drag source object) GUIARGS<1,3,1> = X coordinate in control (drop target) where source was dropped GUIARGS<1,3,2> = Y coordinate in control (drop target) where source was dropped GUIARGS<1,3,3> = character position in the edit control where the drop occurred

1.2.2.4 ATGUICREATELABEL

The ATGUICREATELABEL routine creates a *label control*. A *label control* is used to display text on the form. The user cannot modify the text of a *label control*. By default, a label is displayed on a single line. Change the GPSTYLE property to 1 to create a multi-line label. Normally, if the caption of a *label* includes

the keyboard shortcut prefix "&", then when the user presses the **ALT** key along with the letter following the "&" prefix, the next control after the *label* in the tab order will be activated. To display an ampersand character in the caption, use two ampersands ("&&").

It is possible to disable the "&" shortcut prefix by changing the GPSTYLE property to 2 or 3. Set style to 2 for a single-line "data field" label or 3 for a multi-line "data field" label. With these styles, ampersand characters which are included in the caption text are displayed as-is.

The *value* of a *label control* is the caption text.

Calling syntax:

```
CALL ATGUICREATELABEL(APPID, FORMID, CTRLID, PARENTID, EVENTMASK,
       CAPTION, LEFT, TOP,
       WIDTH, HEIGHT, GUIERRORS, GUISTATE)
```

Input arguments:

APPID	identifies which application control belongs to
FORMID	identifies which form control belongs to
CTRLID	control identifier
PARENTID	identifies a “parent” container control, such as a frame, if any
EVENTMASK	list of events this control responds to
CAPTION	label text; same as GPVALUE property (may include keyboard shortcut using "&" prefix)
LEFT	horizontal position of the control relative to its parent form or frame
TOP	vertical position of the control relative to its parent form or frame
WIDTH	width of the control
HEIGHT	height of the control

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

Properties supported by this item:

GPFORECOLOR	foreground (text) color (default is form’s FORECOLOR)
GPBACKCOLOR	background color (default is parent’s BACKCOLOR)
GPFONTNAME	name of default font (default is form’s FONTNAME)
GPFONTSIZE	font size in points (default is form’s FONTSIZE)
GPFONTBOLD	non-zero to use boldface font
GPFONTITALIC	non-zero to use italic font
GPFONTUNDERLINE	non-zero to use underlined font
GPTRANSPARENT	non-zero if control background is transparent (BACKCOLOR is ignored)

GPENABLED	non-zero if control is enabled
GPVISIBLE	non-zero if control is visible
GPBORDER	0 = no border (default); 1 = flat border; 2 = normal border
GPALIGN	0 = left, 1 = right, 2 = center
GPSTYLE	0 = single line, 1 = multi-line, 2 = data field, 3 = multi-line data field
GPDRAGID	drag source ID (not necessarily the control's CTRLID).
GPDROPIDS	drop target IDs (list of drag source IDs that are allowed to be dropped on this control)
GPDEFVAL	default value of control (also sets current value)
GPVALUE	value of control (may include keyboard shortcut using "&" prefix)

Events supported by this item (sum the desired events to form the EVENTMASK argument):

GECLICK	the user clicked the left mouse button on this control
GECONTEXT	the user clicked the right mouse button on this control
GEDBLCLICK	the user double-clicked the left mouse button on this control
GEDRAGDROP	the user has dropped an acceptable drag source on the control: GUIARGS<1,1> = ID of the drag source that was dropped on the control GUIARGS<1,2,1> = X coordinate within drag source where the drag began GUIARGS<1,2,2> = Y coordinate within drag source where the drag began GUIARGS<1,2,3... > = drag source details (depends on type of drag source object) GUIARGS<1,3,1> = X coordinate in control (drop target) where source was dropped GUIARGS<1,3,2> = Y coordinate in control (drop target) where source was dropped

1.2.2.5 ATGUICREATELIST

The ATGUICREATELIST routine creates a *list control*. The style of control may be single-select, multi-select, check-boxes or drop-down list. A *list control* is used to select one (or more) item(s) from a list of items. A check-box style *list control* is similar to a multi-select *list control*, except that a check-box for each item in the list is displayed at the left edge of the control. Instead of using Shift+Click or Ctrl+Click to select multiple items, check or un-check the check-box.

The *list control* can display multiple columns. The drop-down style *list control* normally shows only the selected item, but when the drop-down button is clicked, the list of items "drops down" so that an item may be selected. Thus the height of a drop-down style *list control* is only the height of a single item. For other styles, the height is the height of the displayed list.

A *list control* can optionally display an icon to the left of the first column. Each item can have a different icon. The *list control* can show grid lines.

The *value* of a *list control* is the one-based index of the selected item or items. The value of zero indicates that no items are selected.

Calling syntax:

```
CALL ATGUICREATELIST(APPID, FORMID, CTRLID, PARENTID, EVENTMASK, STYLE,
       LEFT, TOP,
       WIDTH, HEIGHT, GUIERRORS, GUISTATE)
```

Input arguments:

APPID	identifies which application control belongs to
FORMID	identifies which form control belongs to
PARENTID	identifies a “parent” container control, such as a frame, if any
CTRLID	control identifier
EVENTMASK	list of events this control responds to
STYLE	list control style: 0 = single selection 1 = multiple selection 2 = drop-down list 3 = multiple selection with check-boxes
LEFT	horizontal position of the control relative to its parent form or frame
TOP	vertical position of the control relative to its parent form or frame
WIDTH	width of the control
HEIGHT	height of the control

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

Properties supported by this item:

GPFORECOLOR	foreground (text) color (default is form’s FORECOLOR)
GPBACKCOLOR	background color (default color is WindowBackground)
GPFONTNAME	name of default font (default is form’s FONTNAME)
GPFONTSIZE	font size in points (default is form’s FONTSIZE)
GPFONTBOLD	non-zero to use boldface font
GPFONTITALIC	non-zero to use italic font
GPFONTUNDERLINE	non-zero to use underlined font
GPENABLED	non-zero if control is enabled
GPVISIBLE	non-zero if control is visible
GPHELPID	help topic ID or URL
GPBORDER	0 = no border; 1 = flat border; 2 = normal border (default)
GPREADONLY	non-zero if control is read-only
GPREQUIRED	non-zero to require an item to be selected for this control
GPCOLUMNS	number of columns if multi-column list
GPDATACOL	for multi-column drop-down list, this column is displayed when the drop-down list is closed.

GPGRIDLINES	0 = no grid lines, 1 = horizontal grid lines, 2 = vertical grid lines, 3 = horizontal and vertical grid lines.
GPCOLHEADING	column heading. Separate heading for each column with value marks. An individual column heading can be accessed by ATGUIGETPROP and ATGUISETPROP by specifying a non-zero COL argument.by specifying a non-zero COL argument.
GPCOLDATATYPE	specify one of the enumerated data types for each column. Separate data types for each column with value marks. Note: this property is only used when sorting the list content.
GPCOLALIGN	column alignment: 0 = left (default), 1 = right, 2 = center. Separate alignment settings for each column with value marks. The alignment for an individual column can be accessed by ATGUIGETPROP and ATGUISETPROP by specifying non-zero COL argument.
GPCOLWIDTH	width of column. Separate multiple column widths with value marks. The width of an individual column can be accessed by ATGUIGETPROP and ATGUISETPROP by specifying non-zero COL argument.
GPMAXDROP	maximum number of items to display in the drop-down list (0 = default of 8)
GPITEMS	list contents. Separate rows with value marks, columns with subvalue marks. Individual rows may be accessed by ATGUIGETPROP and ATGUISETPROP by specifying a non-zero ROW argument, and leaving the COL argument zero. Individual fields may be accessed by specifying non-zero COL and ROW arguments.
GPSTYLE	list box style: 0 = single selection 1 = multiple selection 2 = drop-down list 3 = multiple selection with check-boxes
GPICON	icon filename or URL. To set icons for all items, specify zero for COL and ROW and separate the icon filenames with value marks. To set the icon for a specific item, specify the desired row in the ROW argument.
GPICONSIZE	set icon size to 0 for small icons, 1 for large icons.
GPDRAGID	drag source ID (not necessarily the control's CTRLID). When a list control is dropped, GUIARGS<1,2,3> contains the index of the item in the list control where the mouse button was pressed. If the mouse button was not over an item, zero is used.
GPDROPIDS	drop target IDs (list of drag source IDs that are allowed to be dropped on this control)
GPDEFVAL	default value of control (also sets current value).
GPVALUE	index of selected list item or items. Separate multiple selections with value marks.

Events supported by this item (sum the desired events to form the EVENTMASK argument):

GEVALIDATE	the user is attempting to move to another control after changing the selection. Use this event to validate the control's new value. If validation fails, use ATGUIACTIVATE to re-activate this control. GUIARGS<1,1> = ID of the control triggering the event GUIARGS<2> = modified value of the control
GECLICK	the user clicked the left mouse button on an item in the list. A click

	event is also triggered by selecting an item using the cursor keys. GUIARGS<1> is the current value (item index for single selection lists, or list of selected items for multi-selection lists).
GECONTEXT	the user clicked the right mouse button on this control.
GEDBLCLICK	the user double-clicked the left mouse button on an item in the list. GUIARGS<1> is the current value (item index for single selection lists, or list of selected items for multi-selection lists).
GECOLCLICK	the user clicked the left mouse button on the column heading. GUIARGS<1,1> is the column number clicked.
GEACTIVATE	the control has become the active control. GUIARGS<1,1> is the ID of the control being deactivated.
GEDEACTIVATE	the control is no longer the active control. GUIARGS<1,1> is the ID of the control being activated.
GEDRAGDROP	the user has dropped an acceptable drag source on the control: GUIARGS<1,1> = ID of the drag source that was dropped on the control GUIARGS<1,2,1> = X coordinate within drag source where the drag began GUIARGS<1,2,2> = Y coordinate within drag source where the drag began GUIARGS<1,2,3... > = drag source details (depends on type of drag source object) GUIARGS<1,3,1> = X coordinate in control (drop target) where source was dropped GUIARGS<1,3,2> = Y coordinate in control (drop target) where source was dropped GUIARGS<1,3,3> = character position in the edit control where the drop occurred

See also: List Searching

1.2.2.6 ATGUICREATECOMBO

The ATGUICREATECOMBO routine creates a *combo control*. A *combo control* is a combination of *edit* and *list controls*. A *combo control* is used to select an item from a list of items, or enter text directly. The *combo control* can display multiple columns in the drop-down list, and one column may be designated as the “data” column. When a column has been designated as a “data” column, then when the user selects an item in the drop-down list, the contents of the designated “data” column (of the selected item) are copied to the edit portion of the control.

The list portion of a *combo control* can display an icon for each item, and can display grid lines.

The *value* of a *combo control* is the text string of the edit portion of the control.

Calling syntax:

```
CALL ATGUICREATECOMBO(APPID, FORMID, CTRLID, PARENTID, EVENTMASK, LEFT,
       TOP,
       WIDTH, HEIGHT,  GUIERRORS, GUISTATE)
```

Input arguments:

APPID	identifies which application control belongs to
FORMID	identifies which form control belongs to
PARENTID	identifies a “parent” container control, such as a frame, if any
CTRLID	control identifier
EVENTMASK	list of events this control responds to
LEFT	horizontal position of the control relative to its parent form or frame
TOP	vertical position of the control relative to its parent form or frame
WIDTH	width of the control
HEIGHT	height of the control

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

GUISTATE

Properties supported by this item:

GPFORECOLOR	foreground (text) color (default is form’s FORECOLOR)
GPBACKCOLOR	background color (default color is WindowBackground)
GPFONTNAME	name of default font (default is form’s FONTNAME)
GPFONTSIZE	font size in points (default is form’s FONTSIZE)
GPFONTBOLD	non-zero to use boldface font
GPFONTITALIC	non-zero to use italic font
GPFONTUNDERLINE	non-zero to use underlined font
GPENABLED	non-zero if control is enabled
GPVISIBLE	non-zero if control is visible
GPHELPID	help topic ID or URL
GPBORDER	0 = no border; 1 = flat border; 2 = normal border (default)
GPDATATYPE	specify one of the enumerated data types
GPREQUIRED	non-zero to require a value for this control
GPCOLUMNS	number of columns if multi-column list
GPDATACOL	if multi-column list, this is the “data” column
GPGRIDLINES	0 = no grid lines, 1 = horizontal grid lines, 2 = vertical grid lines, 3 = horizontal and vertical grid lines
GPCOLHEADING	drop-down list column heading. Separate headings for each column with value marks. An individual heading can be accessed by ATGUIGETPROP and ATGUISETPROP by specifying a non-zero COL argument.
GPCOLDATATYPE	specify one of the enumerated data types for each column. Separate data types for each column with value marks. Note: this property is only used when sorting the list content.
GPCOLALIGN	column alignment: 0 = left (default), 1 = right, 2 = center. Separate

	alignment settings for each column with value marks. The alignment for an individual column can be accessed by ATGUIGETPROP and ATGUISETPROP by specifying non-zero COL argument.
GPCOLWIDTH	width of column. Separate multiple column widths with value marks. The width of an individual column can be accessed by ATGUIGETPROP and ATGUISETPROP by specifying non-zero COL argument.
GPMAXDROP	maximum number of items to display in the drop-down list (0 = default of 8).
GPSELSTART	starting position of selected text
GPSELLENGTH	length of selected text (zero if no selection)
GPSELRANGE	selection starting position in propval<1,1>, length in propval<1,2>
GPROW	index of the selected row in the drop-down list. If the text in the edit portion does not match any of the items in the list, zero is returned.
GPITEMS	drop-down list contents. Separate rows with value marks, columns with subvalue marks. Individual rows may be accessed by ATGUIGETPROP and ATGUISETPROP by specifying a non-zero ROW argument, and leaving the COL argument zero. Individual fields may be accessed by specifying non-zero COL and ROW arguments.
GPICON	icon filename or URL. To set icons for all items, specify zero for COL and ROW and separate the icon filenames with value marks. To set the icon for a specific item, specify the desired row in the ROW argument.
GPICONSIZE	set icon size to 0 for small icons, 1 for large icons.
GPDRAGID	drag source ID (not necessarily the control's CTRLID). When a combo control is dropped, GUIARGS<1,2,3> contains the character position in the edit portion of the control where the mouse button was pressed.
GPDROPIDS	drop target IDs (list of drag source IDs that are allowed to be dropped on this control)
GPDEFVAL	default value of control (also sets the current value)
GPVALUE	value of control

Events supported by this item (sum the desired events to form the EVENTMASK argument):

GECHANGE	the user changed the control's value – this event is fired for each change (e.g. keystroke).
GEVALIDATE	the user is attempting to move to another control after modifying the control's value. Use this event to validate the control's new value. If validation fails, use ATGUIACTIVATE to re-activate this control. GUIARGS<1,1> = ID of the control triggering the event GUIARGS<2> = modified value of the control
GECLICK	the user clicked the left mouse button on item in the drop-down list, or selected an item using the cursor keys. GUIARGS<1> is the current value of the control.
GECONTEXT	the user clicked the right mouse button on this control.
GEDBLCLICK	the user double-clicked the left mouse button on this control. GUIARGS<1> is the current value of the control.
GECOLCLICK	the user clicked on a column heading; GUIARGS<1> is the column number.
GEACTIVATE	the control has become the active control. GUIARGS<1,1> is the ID of

	the control being deactivated.
GEDEACTIVATE	the control is no longer the active control. GUIARGS<1,1> is the ID of the control being activated.
GEDRAGDROP	the user has dropped an acceptable drag source on the control: GUIARGS<1,1> = ID of the drag source that was dropped on the control GUIARGS<1,2,1> = X coordinate within drag source where the drag began GUIARGS<1,2,2> = Y coordinate within drag source where the drag began GUIARGS<1,2,3... > = drag source details (depends on type of drag source object) GUIARGS<1,3,1> = X coordinate in control (drop target) where source was dropped GUIARGS<1,3,2> = Y coordinate in control (drop target) where source was dropped GUIARGS<1,3,3> = character position in the edit portion of the control where the drop occurred

See also: List Searching

1.2.2.7 ATGUICREATEOPTION

The ATGUICREATEOPTION routine creates an *option group control*. An *option group control* is used to select one of a fixed number of options, similar to “radio buttons”.

You can include an ampersand (&) in the caption text (or in any item text) to use as a "hot key" to activate the control or select the item. When the user presses the letter following the ampersand while holding the **ALT** key, the control will be activated (and the item selected).

The *value* of an *option group control* is index of the selected option. If no option is selected the value is zero.

Calling syntax:

```
CALL ATGUICREATEOPTION(APPID, FORMID, CTRLID, PARENTID, EVENTMASK,
      CAPTION, ITEMS, LEFT, TOP, WIDTH, HEIGHT, GUIERRORS, GUISTATE)
```

Input arguments:

APPID	identifies which application control belongs to
FORMID	identifies which form control belongs to
PARENTID	identifies a “parent” container control, such as a frame, if any
CTRLID	control identifier
EVENTMASK	list of events this control responds to
CAPTION	caption (may include keyboard shortcut using "&" prefix)
ITEMS	text for each option button, separated by value marks. Each item may optionally include a second subvalue specifying whether the item is enabled (1) or disabled (0), and a third subvalue containing

	help hint (tooltip) text for the item. If the second subvalue is missing, the item will be enabled.
LEFT	horizontal position of the control relative to its parent form or frame
TOP	vertical position of the control relative to its parent form or frame
WIDTH	width of the control
HEIGHT	height of the control

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

GUISTATE

Properties supported by this item:

GPFORECOLOR	foreground (text) color (default is form's FORECOLOR)
GPBACKCOLOR	background color (default is parent's BACKCOLOR)
GPFONTNAME	name of default font (default is form's FONTNAME)
GPFONTSIZE	font size in points (default is form's FONTSIZE)
GPFONTBOLD	non-zero to use boldface font
GPFONTITALIC	non-zero to use italic font
GPFONTUNDERLINE	non-zero to use underlined font
GPTRANSPARENT	non-zero if control background is transparent (BACKCOLOR is ignored)
GPENABLED	non-zero if control is enabled. To enable or disable a specific option, call ATGUISETPROP or ATGUIGETPROP with the ROW argument set to the index of the desired option.
GPVISIBLE	non-zero if control is visible
GPCAPTION	caption. To access the caption of a specific option, call ATGUISETPROP or ATGUIGETPROP with the ROW argument set to the index of the desired option.
GPHELPID	help topic ID or URL
GPBORDER	0 = no border; 1 = flat border; 2 = normal border (default)
GPREADONLY	non-zero if control is read-only
GPREQUIRED	non-zero to require an option to be selected for this control
GPARRANGE	0 = down, 1 = across
GPCOLUMNS	0 = automatic, otherwise specify number of columns
GPROWS	0 = automatic, otherwise specify number of rows
GPITEMS	text for each option button separated by value marks. Each item may optionally include a second subvalue specifying whether the item is enabled (1) or disabled (0), and a third subvalue containing help hint (tooltip) text for the item. If the second subvalue is missing, the item will be enabled. An individual item may be accessed by ATGUIGETPROP and ATGUISETPROP by specifying a non-zero ROW argument with the index of the desired option.
GPHINT	hint (or tooltip) text displayed in a balloon window when the cursor

	hovers over the control. Multiple lines of text are separated by subvalue marks. Hint text for individual option items may be accessed by ATGUIGETPROP and ATGUISETPROP by specifying a non-zero ROW argument.
GPDRAGID	drag source ID (not necessarily the control's CTRLID). When an option group control is dropped, GUIARGS<1,2,3> contains the index of the option button where the mouse button was pressed. If the mouse was not over a button, zero is used.
GPDROPIDS	drop target IDs (list of drag source IDs that are allowed to be dropped on this control)
GPDEFVAL	default value of control (also sets the current value)
GPVALUE	value of control (index of selected option or zero for no selected options)

Events supported by this item (sum the desired events to form the EVENTMASK argument):

GEVALIDATE	the user is attempting to move to another control after changing the selected option. Use this event to validate the control's new value. If validation fails, use ATGUIACTIVATE to re-activate this control. GUIARGS<1,1> = ID of the control triggering the event GUIARGS<2> = modified value of the control
GECLICK	the user clicked the left mouse button on this control (a click event is also triggered by selecting an option using the cursor keys). GUIARGS<1> is the new value.
GECONTEXT	the user clicked the right mouse button on this control.
GEACTIVATE	the control has become the active control. GUIARGS<1,1> is the ID of the control being deactivated.
GEDEACTIVATE	the control is no longer the active control. GUIARGS<1,1> is the ID of the control being activated.
GEDRAGDROP	the user has dropped an acceptable drag source on the control: GUIARGS<1,1> = ID of the drag source that was dropped on the control GUIARGS<1,2,1> = X coordinate within drag source where the drag began GUIARGS<1,2,2> = Y coordinate within drag source where the drag began GUIARGS<1,2,3... > = drag source details (depends on type of drag source object) GUIARGS<1,3,1> = X coordinate in control (drop target) where source was dropped GUIARGS<1,3,2> = Y coordinate in control (drop target) where source was dropped GUIARGS<1,3,3> = index of the option button where the mouse button was released. If the mouse was not over a button, zero is used.

1.2.2.8 ATGUICREATECHECK

The ATGUICREATECHECK routine creates a *check box control*. A *check box control* is used to select one of two choices (True/False, On/Off, etc.).

You can include an ampersand (&) in the caption text to use as a "hot key" to activate the control and toggle its state. When the user presses the letter following the ampersand while holding the **ALT** key, the control will be activated and its state will be toggled between checked and unchecked.

The *value* of a *check box control* is zero (un-checked) or one (checked).

Calling syntax:

```
CALL ATGUICREATECHECK(APPID, FORMID, CTRLID, PARENTID, EVENTMASK,
       CAPTION, LEFT, TOP,
       WIDTH, HEIGHT, GUIERRORS, GUISTATE)
```

Input arguments:

APPID	identifies which application control belongs to
FORMID	identifies which form control belongs to
PARENTID	identifies a “parent” container control, such as a frame, if any
CTRLID	control identifier
EVENTMASK	list of events this control responds to
CAPTION	text. Multiple lines are separated by value marks.
LEFT	horizontal position of the control relative to its parent form or frame
TOP	vertical position of the control relative to its parent form or frame
WIDTH	width of the control
HEIGHT	height of the control

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

Properties supported by this item:

GPFORECOLOR	foreground (text) color (default is form’s FORECOLOR)
GPBACKCOLOR	background color (default is parent’s BACKCOLOR)
GPFONTNAME	name of default font (default is form’s FONTNAME)
GPFONTSIZE	font size in points (default is form’s FONTSIZE)
GPFONTBOLD	non-zero to use boldface font
GPFONTITALIC	non-zero to use italic font
GPFONTUNDERLINE	non-zero to use underlined font
GPTRANSPARENT	non-zero if control background is transparent (BACKCOLOR is ignored)

GPENABLED	non-zero if control is enabled
GPVISIBLE	non-zero if control is visible
GPALIGN	0 = left, 1 = right, 2 = center
GPCAPTION	caption text. Multiple lines are separated by value marks.
GPHELPID	help topic ID or URL.
GPDRAGID	drag source ID (not necessarily the control's CTRLID).
GPDROPIDS	drop target IDs (list of drag source IDs that are allowed to be dropped on this control)
GPDEFVAL	default value of control (also sets the current value).
GPVALUE	value of control (checkbox state, 0 or 1).

Events supported by this item (sum the desired events to form the EVENTMASK argument):

GEVALIDATE	the user is attempting to move to another control after toggling the state of this control. Use this event to validate the control's new value. If validation fails, use ATGUIACTIVATE to re-activate this control. GUIARGS<1,1> = ID of the control triggering the event GUIARGS<2> = modified value of the control
GECLICK	the user toggled the state of this control by clicking the left mouse button, by pressing the spacebar when the control is active, or by using a keyboard shortcut. GUIARGS<1,1> contains the new control value.
GECONTEXT	the user clicked the right mouse button on this control.
GEACTIVATE	the control has become the active control. GUIARGS<1,1> is the ID of the control being deactivated.
GEDEACTIVATE	the control is no longer the active control. GUIARGS<1,1> is the ID of the control being activated.
GEDRAGDROP	the user has dropped an acceptable drag source on the control: GUIARGS<1,1> = ID of the drag source that was dropped on the control GUIARGS<1,2,1> = X coordinate within drag source where the drag began GUIARGS<1,2,2> = Y coordinate within drag source where the drag began GUIARGS<1,2,3... > = drag source details (depends on type of drag source object) GUIARGS<1,3,1> = X coordinate in control (drop target) where source was dropped GUIARGS<1,3,2> = Y coordinate in control (drop target) where source was dropped

1.2.2.9 ATGUICREATEBUTTON

The ATGUICREATEBUTTON routine creates a *command button control*. A *command button control* is used to trigger an action when the user “clicks” the button. To detect the “click”, the event mask must contain GECLICK.

You can include an ampersand (&) in the caption text to use as a “hot key” to activate the button and fire the Click event. When the user presses the letter following the ampersand while holding the **ALT** key,

the control will be activated and the Click event will fire.

When a *command button control* is active (has focus), pressing the **SPACE BAR** will fire the Click event. If the GPRTNEQTAB application property is not set to 1, pressing the **ENTER** key when a *command button control* is active will also fire the Click event.

Two special “styles” are available for command buttons: OK (1) and Cancel (2). Only one button on a form can be set to one of these styles. If a button currently has one of these styles, and you set a different button to the same style, the original button loses the special style. If a button has the OK style, and the GPRTNEQTAB application property is not 1, pressing the **ENTER** key when any other control is active causes the OK button to be activated and fire a Click event. If a button has the Cancel style, pressing the **ESC** key will activate the button and fire a Click event. The Cancel button does not cause local data type validation or enforce the Required property of the control that had focus before the Cancel button was clicked.

A third special style,"toolbar button" (3), is also available. Command buttons using this style appear like toolbar buttons. Their border is invisible until the mouse is positioned over the button. Buttons with this style *do not get activated* (they cannot get focus). Because these buttons cannot be activated, clicking on them will *not cause the active control to fire its Validate or Deactivate events.*

Graphical buttons can be created by specifying a border style of "none" and specifying a picture for the button. Pictures with transparency, such as PNG files with alpha channel, are supported for graphical buttons.

The *command button control* does not have a *value*.

Calling syntax:

```
CALL ATGUICREATEBUTTON(APPID, FORMID, CTRLID, PARENTID,
       EVENTMASK, CAPTION, LEFT, TOP, WIDTH, HEIGHT,
       GUIERRORS, GUISTATE)
```

Input arguments:

APPID	identifies which application control belongs to
FORMID	identifies which form control belongs to
PARENTID	identifies a “parent” container control, such as a frame, if any
CTRLID	control identifier
EVENTMASK	list of events this control responds to (must include GECLICK)
CAPTION	button caption. Multiple lines are separated by value marks.
LEFT	horizontal position of the control relative to its parent form or frame
TOP	vertical position of the control relative to its parent form or frame
WIDTH	width of the control
HEIGHT	height of the control

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

GUISTATE

Properties supported by this item:

GPFORECOLOR	foreground (text) color (default is form's FORECOLOR)
GPBACKCOLOR	background color (default is form's 3DFace). Note: if you use a custom background color, the button's appearance will look different from other buttons which use the default background color. Buttons using the default background color will render like other command buttons used by Windows.
GPFONTNAME	name of default font (default is form's FONTNAME)
GPFONTSIZE	font size in points (default is form's FONTSIZE)
GPFONTBOLD	non-zero to use boldface font
GPFONTITALIC	non-zero to use italic font
GPFONTUNDERLINE	non-zero to use underlined font
GPENABLED	non-zero if control is enabled
GPVISIBLE	non-zero if control is visible
GPCAPTION	button caption. Multiple lines are separated by value marks.
GPSTYLE	command button style: 0 = normal 1 = OK button 2 = Cancel button 3 = toolbar button
GPBORDER	2 = normal border (default), 0 = borderless (for graphical buttons)
GPHELPID	help topic ID or URL
GPPICTURE	optional picture filename or URL for graphical buttons (do not use GPCAPTION for graphical buttons)
GPICON	icon filename or URL
GPICONSIZE	set icon size to 0 for small icons, 1 for large icons.
GPICONALIGN	icon alignment: 0 = left 1 = right 2 = center 3 = top 4 = bottom
GPDRAGID	drag source ID (not necessarily the control's CTRLID).
GPDROPIDS	drop target IDs (list of drag source IDs that are allowed to be dropped on this control)

Events supported by this item (sum the desired events to form the EVENTMASK argument):

GECLICK	the user clicked the left mouse button on this control, pressed the spacebar when the control is active, or activated the button using a keyboard shortcut.
GECONTEXT	the user clicked the right mouse button on this control.
GEACTIVATE	the control has become the active control. GUIARGS<1,1> is the ID of the control being deactivated. This event is not supported by toolbar-style buttons.
GEDEACTIVATE	the control is no longer the active control. GUIARGS<1,1> is the ID of

	the control being activated. This event is not supported by toolbar-style buttons.
GESTATUS	the GESTATUS event is fired when the mouse button is pressed and again when it is released. This event might be used where you need an "auto repeat" function. GUIARGS<1,1,1> = button number: 1 = left, 2 = right, 3 = middle GUIARGS<1,1,2> = shift state: 0 = none, 1 = **SHIFT**, 2 = **CTRL**, 3 = **SHIFT+CTRL** GUIARGS<1,1,3> = button state: 0 = released, 1 = depressed GUIARGS<1,1,4> = X coordinate of mouse when button was pressed or released GUIARGS<1,1,5> = Y coordinate of mouse when button was pressed or released Note that a GECONTEXT event takes precedence over the GESTATUS event for right-button clicks. X & Y are relative to the upper-left corner of the command button.
GEDRAGDROP	the user has dropped an acceptable drag source on the control: GUIARGS<1,1> = ID of the drag source that was dropped on the control GUIARGS<1,2,1> = X coordinate within drag source where the drag began GUIARGS<1,2,2> = Y coordinate within drag source where the drag began GUIARGS<1,2,3... > = drag source details (depends on type of drag source object) GUIARGS<1,3,1> = X coordinate in control (drop target) where source was dropped GUIARGS<1,3,2> = Y coordinate in control (drop target) where source was dropped

1.2.2.10 ATGUICREATEGRID

The ATGUICREATEGRID routine creates a *grid control*. A *grid control* is two-dimensional spreadsheet-like control used to display and enter multiple rows of correlated data. This control is useful for displaying and editing a correlated set of multi-valued fields.

Each column of a *grid control* can contain label fields (read-only text), editable text fields, check boxes, drop-down lists or drop-down combos. Any column can be designated "read only", and columns may be resizable at runtime. Label columns may include a clickable button (ellipsis button, or specify an icon). Label columns may contain an icon to the left of the text.

The lists displayed for drop-down list and drop-down combo columns may themselves contain multiple columns. With multi-column lists, one column of the list may be designated as the “data” column.

A *grid control* may be editable or protected.

Users navigate a grid by using the cursor keys, **TAB** key (and **ENTER** key if the GPRTNEQTAB application property is not zero). If a grid is editable, normal cursor movements bypass any label or read-only columns. You can type data directly into the active cell (if it's a text or combo field), or you can press the **F2** key to edit data in the cell.

The *value* of a *grid control* is a two-dimensional array of the values of the cells of the grid. Values in check-box columns are either one (checked) or zero (un-checked).

Calling syntax:

```
CALL ATGUICREATEGRID(APPID, FORMID, CTRLID, PARENTID, EVENTMASK, STYLE,
       COLUMNS, COLHEADING, LEFT, TOP, WIDTH, HEIGHT, GUIERRORS,
       GUISTATE)
```

Input arguments:

APPID	identifies which application control belongs to
FORMID	identifies which form control belongs to
PARENTID	identifies a “parent” container control, such as a frame, if any
CTRLID	control identifier
EVENTMASK	list of events this control responds to
STYLE	0 = protected, 1 = editable
COLUMNS	number of columns in the grid
COLHEADING	list (column) heading (separate column headings with value marks)
LEFT	horizontal position of the control relative to its parent form or frame
TOP	vertical position of the control relative to its parent form or frame
WIDTH	width of the control
HEIGHT	height of the control

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

Properties supported by this item:

GPFORECOLOR	foreground (text) color (default is form’s FORECOLOR). Individual column, row or cell colors may be updated by specifying non-zero values for the COL and ROW parameters.
GPBACKCOLOR	background color (default color is WindowBackground). Individual column, row or cell colors may be updated by specifying non-zero values for the COL and ROW parameters.
GPALTCOLOR	alternate row background color (default is the normal background color). All even row numbers are displayed using the alternate background color; odd row numbers are displayed using the normal background color.
GPFONTNAME	name of default font (default is form’s FONTNAME)
GPFONTSIZE	font size in points (default is form’s FONTSIZE)
GPFONTBOLD	non-zero to use boldface font
GPFONTITALIC	non-zero to use italic font

GPFONTUNDERLINE	non-zero to use underlined font
GPENABLED	non-zero if control is enabled
GPVISIBLE	non-zero if control is visible
GPHELPID	help topic ID or URL
GPBORDER	0 = no border; 1 = flat border; 2 = normal border (default)
GPGRIDLINES	0 = no grid lines, 1 = horizontal grid lines, 2 = vertical grid lines, 3 = horizontal and vertical grid lines
GPSTYLE	0 = normal, 1 = auto extend (adds new rows to editable grid automatically)
GPCOLUMNS	number of columns in the grid
GPROWS	number of data rows in the grid
GPFIXEDCOLS	number of non-scrolling columns in the grid (zero to scroll all columns)
GPCOLHEADING	grid column heading. Separate column headings with value marks. An individual heading can be accessed by ATGUIGETPROP and ATGUISETPROP by specifying a non-zero COL argument.
GPCOLDATATYPE	specify one of the enumerated data types for each column. Separate data types for each column with value marks.
GPCOLFIELDTYPE	column type: 0 = label 1 = edit 2 = check box 3 = drop-down list 4 = drop-down combo 5 = label with ellipsis button 6 = label with icon displayed to the left of the label text 7 = label with custom button (assign icon by setting GPICON property) Note: only label, edit and check box are supported for non-editable grids. Cell buttons fire the GEBTNCLICK event when clicked.
GPCOLALIGN	column alignment: 0 = left (default), 1 = right, 2 = center . Separate each column's alignment with value marks. The alignment for an individual column can be accessed by ATGUIGETPROP and ATGUISETPROP by specifying non-zero COL argument.
GPCOLWIDTH	width of column. Separate each column's width with value marks. The width for an individual column can be accessed by ATGUIGETPROP and ATGUISETPROP by specifying non-zero COL argument.
GPDATACOL	for each grid column, if the column has a drop-down list with multiple columns, this is the "data" column of the drop-down list. Separate each column's setting with value marks. Columns without a drop-down list are ignored. The "data" column for an individual grid column can be accessed by ATGUIGETPROP and ATGUISETPROP by specifying non-zero COL argument.
GPCOLITEMS	for grid columns with drop-down lists, specify the list items for each column. Separate individual list items with subvalue marks; separate columns within a list item with vertical bars (|), separate each grid column's list with value marks. Columns without drop-down lists are ignored. The list for an individual column can be accessed by ATGUIGETPROP and ATGUISETPROP by specifying non-zero COL argument.

GPREQUIRED	non-zero to require data in a column. Separate each column's setting with value marks. The setting for an individual column can be accessed by ATGUIGETPROP and ATGUISETPROP by specifying non-zero COL argument.
GPMAXLEN	for edit and combo columns, specifies the maximum number of characters to accept, or zero for no limit. Separate each column's setting with value marks. The setting for an individual column can be accessed by ATGUIGETPROP and ATGUISETPROP by specifying non-zero COL argument.
GPCOLSIZABLE	set to 1 if a column is resizable at runtime, or zero if the column cannot be resized. When the user resizes a column at runtime, a Resize event is fired. Separate each column's setting with value marks. The setting for an individual column can be accessed by ATGUIGETPROP and ATGUISETPROP by specifying non-zero COL argument.
GPREADONLY	set to 1 if a column is read-only, or zero if it can be modified. Separate each column's setting with value marks. The setting for an individual column can be accessed by ATGUIGETPROP and ATGUISETPROP by specifying non-zero COL argument.
GPCOLHINT	column hint (tooltip) text displayed in a balloon window when the cursor hovers over a column heading. Separate hint text for each column with value marks, multiple lines of text are separated with subvalue marks. An individual column's hint can be accessed by ATGUIGETPROP and ATGUISETPROP by specifying a non-zero COL argument.
GPCOLTABSTOP	set to 1 if a column should not be skipped when using the cursor to navigate the grid. Set to 0 to skip the column. This property overrides the default behavior of skipping label and read-only columns. Separate each column's setting with value marks. The setting for an individual column can be accessed by ATGUIGETPROP and ATGUISETPROP by specifying non-zero COL argument.
GPHINT	hint (or tooltip) text displayed in a balloon window when the cursor hovers over a cell in the grid. Multiple lines of text are separated by subvalue marks. An individual cell's hint can be accessed by ATGUIGETPROP and ATGUISETPROP by specifying a non-zero COL and ROW arguments.
GPICON	sets the icon to display in "label+icon" or "label+button" columns. An individual cell's icon can be accessed by ATGUIGETPROP and ATGUISETPROP by specifying a non-zero COL and ROW arguments.
GPCOLUMN	current grid column. Setting this property activates the specified cell and scrolls it into view if it is not currently visible.
GPROW	current grid row. Setting this property activates the specified cell and scrolls it into view if it is not currently visible.
GPWORDWRAP	non-zero to enable word wrapping within grid cells. *Note: when word wrapping is enabled and the grid cannot display all of the columns, text in the last displayed column may not wrap properly.*
GPSELSTART	returns the upper-left corner of the selected cells in the grid. propval <1,1> is the column, propval<1,2> is the row. Zero if no selection.
GPSELLENGTH	returns the extent of the selection in the grid: propval<1,1> is the number of columns, propval<1,2> is the number of rows. Zero if no selection.

GPSELRANGE	returns the coordinates of the selection in the grid: propval<1,1> is the upper-left column, propval<1,2> is the upper-left row, propval <1,3> is the lower-right column and propval<1,4> is the lower-right row.
GPFOCUSSTYLE	active grid cell focus style: 0 = no focus indicator 1 = light dotted line 2 = heavy dotted line 3 = solid line (default)
GPPASTEMODE	controls whether a "paste" operation affects a single cell (0) or multiple cells (1). Multi-cell pasting from Excel is permitted when this property is set to 1 (the default is 0).
GPDRAGMODE	controls grid behavior when dragging within a grid: 0 = change active cell (default) 1 = use drag-and-drop 2 = extend selection 3 = ignore dragging
GPDRAGID	drag source ID (not necessarily the control's CTRLID). When a grid cell is dropped, GUIARGS<1,2,3> contains the cell column and GUIARGS<1,2,4> contains the cell row where the mouse button was pressed. If the mouse was over a column heading, the row is zero. If the mouse was not over a cell, GUIARGS<1,2,3> is null.
GPDROPIDS	drop target IDs (list of drag source IDs that are allowed to be dropped on this control)
GPDEFVAL	default value of control (also sets the current value).
GPVALUE	value of the grid. Rows are separated by value marks, columns in each row are separated by subvalue marks.

Events supported by this item (sum the desired events to form the EVENTMASK argument):

GECLICK	the user clicked the left mouse button on this control. GUIARGS<1,1> is the column number and GUIARGS<1,2> is the row number of the cell clicked. If the user clicked in the unused area after the last row, the row number is -1.
GEDBLCLICK	the user double-clicked the left mouse button on this control. GUIARGS<1,1> is the column number and GUIARGS<1,2> is the row number of the cell clicked. If the user clicked in the heading row, the row number is 0; if he clicked in the unused area after the last row, the row number is -1.
GECONTEXT	the user clicked the right mouse button on this control. GUIARGS<1,1> is the column number and GUIARGS<1,2> is the row number of the cell clicked. If the user clicked in the heading row, the row number is 0; if he clicked in the unused area after the last row, the row number is -1.
GECOLCLICK	the user clicked the left mouse button on the column heading. GUIARGS<1,1> is the column number clicked.
GEBTNCLICK (formerly GEELLIPSIS)	the user clicked the cell button. GUIARGS<1,1> is the column number and GUIARGS<1,2> is the row number of the cell where the button was clicked.
GERESIZE	the user has resized a column by dragging the column separator in the heading. The new column width is GUIARGS<1,1> and column number is GUIARGS<1,3>.

GECHANGE	the user has finished editing data in a text field, selected an item from a drop-down list, or toggled the state of a checkbox. GUIARGS<1,1> = column number of changed cell GUIARGS<1,2> = row number of changed cell GUIARGS<2> = new value of cell
GEVALIDATE	the user is attempting to move to another control after modifying the at least one cell in the grid. Use this event to validate the grid's new value. If validation fails use ATGUIACTIVATE to re-activate this control. GUIARGS<1,1> = ID of the control triggering the event GUIARGS<2> = modified value of the grid. Only editable grids fire this event.
GEVALIDATECELL	the user finished editing a cell and moved to a different cell (or activated a different control). GUIARGS<1,1> = column number of cell being validated GUIARGS<1,2> = row number of cell being validated GUIARGS<1,3> = column number of next cell GUIARGS<1,4> = row number of next cell GUIARGS<1,5> = ID of next control GUIARGS<2> = new cell value Only editable grids fire this event. If the user is attempting to move to another cell in the same grid, GUIARGS<1,3> and GUIARGS<1,4> specify that cell. If the user is attempting to move to a different control, GUIARGS<1,3> and GUIARGS<1,4> are null and GUIARGS<1,5> specifies the ID of the control triggering the event.
GEVALIDATEROW	the user has changed data in at least one cell of the previously active row and moved to a different row (or activated a different control). GUIARGS<1,1> = row number being validated GUIARGS<2> = values of all of the row's cells separated by subvalue marks. Only editable grids fire this event.
GEACTIVATE	the control has become the active control. GUIARGS<1,1> is the ID of the control being deactivated.
GEACTIVATECELL	the user has moved to a new cell (or the grid has just become the active control). GUIARGS<1,1> = new column number GUIARGS<1,2> = new row number GUIARGS<1,3> = previous column number GUIARGS<1,4> = previous row number GUIARGS<1,5> = ID of the previously active control if the grid has just been activated (previous column & row will be null in this case); if the grid was already active, this value is null.
GEACTIVATEROW	the user has moved to a new row (or the grid has just become the active control). GUIARGS<1,1> = new row number GUIARGS<1,2> = previous row number GUIARGS<1,5> = ID of the previously active control if the grid has just been activated (previous row will be null in this case); if the grid was already active, this value is null.
GEDEACTIVATE	the control is no longer the active control. GUIARGS<1,1> is the ID of the control being activated.
GEDEACTIVATECELL	the user has moved to a new cell (or the grid has just been deactivated). GUIARGS<1,1> = previous column number

	GUIARGS<1,2> = previous row number GUIARGS<1,3> = new column number GUIARGS<1,4> = new row number GUIARGS<1,5> = ID of the newly activated control if the grid has just been deactivated (new column & row will be null in this case); if the grid was already active, this value is null.
GEDEACTIVATEROW	the user has moved to a new row (or the grid has just been deactivated). GUIARGS<1,1> = previous row number GUIARGS<1,2> = new row number GUIARGS<1,3> = ID of the newly activated control if the grid has just been deactivated (new row will be null in this case); if the grid was already active, this value is null.
GEDRAGDROP	the user has dropped an acceptable drag source on the control: GUIARGS<1,1> = ID of the drag source that was dropped on the control GUIARGS<1,2,1> = X coordinate within drag source where the drag began GUIARGS<1,2,2> = Y coordinate within drag source where the drag began GUIARGS<1,2,3... > = drag source details (depends on type of drag source object) GUIARGS<1,3,1> = X coordinate in control (drop target) where source was dropped GUIARGS<1,3,2> = Y coordinate in control (drop target) where source was dropped GUIARGS<1,3,3> = target cell column, or null if not over a cell or heading GUIARGS<1,3,4> = target cell row, or zero if over a column heading

See also: List Searching

1.2.2.11 ATGUICREATEPICTURE

The ATGUICREATEPICTURE routine creates a *picture control*. A *picture control* can be used to display an image, as a button, or as a container for other controls. To use a *picture control* as a button, the event mask must contain GECLICK The *picture control's* value is the picture file name or URL. When a *picture control* is used as a container for other controls, specify the CTLID of the *picture control* as the PARENTID argument when creating the contained controls. The image in a *picture control* can be displayed without scaling, scaled to fit the control, or the control may be resized to fit the image.

Calling syntax:

```
CALL ATGUICREATEPICTURE(APPID, FORMID, CTRLID, PARENTID, EVENTMASK,
      STYLE, FILENAME,
      LEFT, TOP, WIDTH, HEIGHT, GUIERRORS, GUISTATE)
```

Input arguments:

APPID	identifies which application control belongs to
FORMID	identifies which form control belongs to

CTRLID	control identifier
PARENTID	identifies a “parent” container control, such as a frame, if any
EVENTMASK	list of events this control responds to (must include GECLICK)
STYLE	picture scaling mode: 0 = no scaling 1 = scale image to fit control preserving image aspect ratio 2 = scale image to fit control 3 = resize control to fit image
FILENAME	filename or URL of picture. Picture formats supported include BMP, JPEG, GIF, PCD, PCX, PICT, PNG, PSD, TARGA, TIFF, WBMP, XBM, XPM and Windows Metafile.
LEFT	horizontal position of the control relative to its parent form or frame
TOP	vertical position of the control relative to its parent form or frame
WIDTH	width of the control
HEIGHT	height of the control

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

GUISTATE

Properties supported by this item:

GPENABLED	non-zero if control is enabled
GPVISIBLE	non-zero if control is visible
GPBORDER	0 = no border; 1 = flat border; 2 = normal border (default)
GPSTYLE	picture scaling mode: 0 = no scaling 1 = scale image to fit control preserving image aspect ratio 2 = scale image to fit control 3 = resize control to fit image
GPPICTURE	filename or URL of picture. Picture formats supported include BMP, JPEG, GIF, PCD, PCX, PICT, PNG, PSD, TARGA, TIFF, WBMP, XBM, XPM and Windows Metafile.
GPSTATUS	returns status information about the control: if COL = 2, returns a multi-valued list of child objects (controls which have the picture object as their parent).
GPDRAGID	drag source ID (not necessarily the control's CTRLID).
GPDROPIDS	drop target IDs (list of drag source IDs that are allowed to be dropped on this control)

Events supported by this item (sum the desired events to form the EVENTMASK argument):

GECLICK	the user clicked the left mouse button on this control
GEDBLCLICK	the user double-clicked the left mouse button on this control

GECONTEXT	the user clicked the right mouse button on this control
GEDRAGDROP	the user has dropped an acceptable drag source on the control: GUIARGS<1,1> = ID of the drag source that was dropped on the control GUIARGS<1,2,1> = X coordinate within drag source where the drag began GUIARGS<1,2,2> = Y coordinate within drag source where the drag began GUIARGS<1,2,3... > = drag source details (depends on type of drag source object) GUIARGS<1,3,1> = X coordinate in control (drop target) where source was dropped GUIARGS<1,3,2> = Y coordinate in control (drop target) where source was dropped

1.2.2.12 ATGUICREATEFRAME2

The ATGUICREATEFRAME2 routine creates a *frame control*. A *frame control* is used as a container for other controls. When creating the contained controls, specify the CTLID of the *frame control* as the PARENTID argument. If the caption of a *frame* includes the keyboard shortcut prefix "&", then when the user presses the **ALT** key along with the letter following the "&" prefix, the first activatable control in the *frame* will be activated.

Calling syntax:

```
CALL ATGUICREATEFRAME2(APPID, FORMID, CTRLID, PARENTID, EVENTMASK,
       CAPTION, STYLE, LEFT, TOP, WIDTH, HEIGHT, GUIERRORS, GUISTATE)
```

Input arguments:

APPID	identifies which application control belongs to
FORMID	identifies which form control belongs to
PARENTID	identifies a “parent” container control, such as a frame, if any
CTRLID	control identifier
EVENTMASK	list of events this control responds to (must include GECLICK)
CAPTION	caption text (may include keyboard shortcut using "&" prefix)
LEFT	horizontal position of the control relative to its parent form or frame
STYLE	0 = normal frame, 1 = frame without caption (allows slightly more space inside the frame for placement of other controls)
TOP	vertical position of the control relative to its parent form or frame
WIDTH	width of the control
HEIGHT	height of the control

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

GUISTATE

Properties supported by this item:

GPFORECOLOR	foreground (text) color (default is form's FORECOLOR)
GPBACKCOLOR	background color (default is parent's BACKCOLOR)
GPFONTNAME	name of default font (default is form's FONTNAME)
GPFONTSIZE	font size in points (default is form's FONTSIZE)
GPFONTBOLD	non-zero to use boldface font
GPFONTITALIC	non-zero to use italic font
GPUNDERLINE	non-zero to use underlined font
GPTRANSPARENT	non-zero if control background is transparent (BACKCOLOR is ignored)
GPENABLED	non-zero if control is enabled
GPVISIBLE	non-zero if control is visible
GPBORDER	0 = no border; 1 = flat border; 2 = normal border (default)
GPCAPTION	caption text
GPSTYLE	0 = normal frame, 1 = frame without caption (allows slightly more space inside the frame for placement of other controls)
GPSTATUS	returns status information about the form: if COL = 2, returns a multi-valued list of child objects (controls which have the frame as their parent).
GPDRAGID	drag source ID (not necessarily the control's CTRLID).
GPDROPIDS	drop target IDs (list of drag source IDs that are allowed to be dropped on this control)

Events supported by this item (sum the desired events to form the EVENTMASK argument):

GECLICK	the user clicked the left mouse button on this control
GEDBLCLICK	the user double-clicked the left mouse button on this control
GECONTEXT	the user clicked the right mouse button on this control
GEDRAGDROP	the user has dropped an acceptable drag source on the control: GUIARGS<1,1> = ID of the drag source that was dropped on the control GUIARGS<1,2,1> = X coordinate within drag source where the drag began GUIARGS<1,2,2> = Y coordinate within drag source where the drag began GUIARGS<1,2,3... > = drag source details (depends on type of drag source object) GUIARGS<1,3,1> = X coordinate in control (drop target) where source was dropped GUIARGS<1,3,2> = Y coordinate in control (drop target) where source was dropped

Note: ATGUICREATEFRAME2 supersedes the ATGUICREATEFRAME routine.

1.2.2.13 ATGUICREATETABGRP

The ATGUICREATETABGRP routine creates a *tab group control*. A *tab group control* is used as a container for *tab controls.* Each *tab control* contained within a *tab group control* is displayed as an "index tab" in the *tab group control.* When creating the contained *tab controls*, specify the CTLID of the *tab group control* as the PARENTID argument. The *tab group control* may also contain other controls besides *tabs*. Any other controls created within a *tab group control* are displayed on every *tab*. This is useful for displaying an OK or Cancel button on each *tab,* without requiring a separate button on each. The value of the *tab group* is the index of the currently selected *tab* (the first tab index is 1). You can change the currently selected tab by updating the GPVALUE property of the *tab group.*

Calling syntax:

```
CALL ATGUICREATETABGRP(APPID, FORMID, CTRLID, PARENTID, EVENTMASK, LEFT,
       TOP,
       WIDTH, HEIGHT, GUIERRORS, GUISTATE)
```

Input arguments:

APPID	identifies which application control belongs to
FORMID	identifies which form control belongs to
PARENTID	identifies a “parent” container control, such as a frame, if any
CTRLID	control identifier
EVENTMASK	list of events this control responds to
LEFT	horizontal position of the control relative to its parent form or frame
TOP	vertical position of the control relative to its parent form or frame
WIDTH	width of the control
HEIGHT	height of the control

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

Properties supported by this item:

GPFORECOLOR	foreground (text) color (default is form’s FORECOLOR)
GPBACKCOLOR	background color (default color is form's BACKCOLOR)
GPFONTNAME	name of default font (default is form’s FONTNAME)
GPFONTSIZE	font size in points (default is form’s FONTSIZE)
GPFONTBOLD	non-zero to use boldface font
GPFONTITALIC	non-zero to use italic font
GPFONTUNDERLINE	non-zero to use underlined font
GPENABLED	non-zero if control is enabled
GPVISIBLE	non-zero if control is visible

GPSTYLE	0 = single tab row, 1 = stacked tabs
GPSTATUS	returns status information about the form: if COL = 2, returns a multi-valued list of child objects (controls which have the tab group as their parent).
GPDRAGID	drag source ID (not necessarily the control's CTRLID). When a tab is dropped, GUIARGS<1,2,3> contains the index of the tab where the mouse button was pressed. If the mouse was not over a tab, zero is used.
GPDROPIDS	drop target IDs (list of drag source IDs that are allowed to be dropped on this control)
GPDEFVAL	default value of control (also sets the current value)
GPVALUE	value of control (index of currently selected tab).

Events supported by this item (sum the desired events to form the EVENTMASK argument):

GECLICK	The selected tab has changed, either due the user clicking on a different tab, or pressing a hotkey to activate a different tab. The index of selected tab is GUIARGS<1,1>.
GECONTEXT	the user clicked the right mouse button on this control.
GEACTIVATE	the control has become the active control. GUIARGS<1,1> is the ID of the control being deactivated.
GEDEACTIVATE	the control is no longer the active control. GUIARGS<1,1> is the ID of the control being activated.
GEDRAGDROP	the user has dropped an acceptable drag source on the control: GUIARGS<1,1> = ID of the drag source that was dropped on the control GUIARGS<1,2,1> = X coordinate within drag source where the drag began GUIARGS<1,2,2> = Y coordinate within drag source where the drag began GUIARGS<1,2,3... > = drag source details (depends on type of drag source object) GUIARGS<1,3,1> = X coordinate in control (drop target) where source was dropped GUIARGS<1,3,2> = Y coordinate in control (drop target) where source was dropped GUIARGS<1,3,3> = index of the tab where the mouse button was released. If the mouse was not over a tab, zero is used.

1.2.2.14 ATGUICREATETAB

The ATGUICREATETAB routine creates a *tab control*. A *tab control* is used as a container for other controls and must be contained in a *tab group control*. When creating a *tab control*, specify the ID of a *tab group control* as the PARENTID argument. When creating the contained controls, specify the CTLID of the *tab control* as the PARENTID argument. The containing *tab group control* defines the size of the *tab control*, thus no size or position parameters are required when creating a *tab control*. To select a *tab control* set the GPVALUE property of the containing *tab group control* to the index of the desired *tab control,* or call ATGUIACTIVATE and specify the CTLID of the *tab control* to be selected.

You can include an ampersand (&) in the tab caption to use as a "hot key" to activate that tab. When

the user presses the letter following the ampersand while holding the **ALT** key, the tab will be activated.

Calling syntax:

```
CALL ATGUICREATETAB(APPID, FORMID, CTRLID, PARENTID, EVENTMASK, CAPTION,
      GUIERRORS, GUISTATE)
```

Input arguments:

APPID	identifies which application control belongs to
FORMID	identifies which form control belongs to
PARENTID	identifies the “parent” TabGroup control
CTRLID	control identifier
EVENTMASK	list of events this control responds to
CAPTION	tab caption text

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

Properties supported by this item:

GPFORECOLOR	foreground (text) color (default is parent TabGroup FORECOLOR)
GPBACKCOLOR	background color (default is parent TabGroup BACKCOLOR)
GPENABLED	non-zero if control is enabled
GPVISIBLE	non-zero if control is visible
GPCAPTION	caption text
GPSTATUS	returns status information about the form: if COL = 2, returns a multi-valued list of child objects (controls which have the tab as their parent).
GPICON	icon filename or URL
GPDROPIDS	drop target IDs (list of drag source IDs that are allowed to be dropped on this control)

Events supported by this item (sum the desired events to form the EVENTMASK argument):

GECLICK	The selected tab has changed, either due the user clicking on a different tab, or pressing a hotkey to activate a different tab. The index of selected tab is GUIARGS<1,1>.
GECONTEXT	the user clicked the right mouse button on this control.
GEDRAGDROP	the user has dropped an acceptable drag source on the control: GUIARGS<1,1> = ID of the drag source that was dropped on the control GUIARGS<1,2,1> = X coordinate within drag source where the drag began

	GUIARGS<1,2,2> = Y coordinate within drag source where the drag began GUIARGS<1,2,3... > = drag source details (depends on type of drag source object) GUIARGS<1,3,1> = X coordinate in control (drop target) where source was dropped GUIARGS<1,3,2> = Y coordinate in control (drop target) where source was dropped

1.2.2.15 ATGUICREATETREE

The ATGUICREATETREE routine creates a *tree control*. A *tree* control is used to display a hierarchical representation of data, similar to Windows Explorer. The structure consists of one or more top-level nodes, which may contain one or more child nodes. The *tree* can display a "plus" or "minus" sign for nodes that have "children". Clicking on the sign causes the node to expand or collapse. Each node contains a node ID, a description, and optionally a large or small icon.

The value of a *tree* control is the "path" for the selected node, which consists of each node ID from the top-level node to the selected node separated by a backslash (\) character.

Calling syntax:

```
CALL ATGUICREATETREE(APPID, FORMID, CTRLID, PARENTID, EVENTMASK, STYLE,
       LEFT, TOP,
       WIDTH, HEIGHT, GUIERRORS, GUISTATE)
```

Input arguments:

APPID	identifies which application control belongs to
FORMID	identifies which form control belongs to
PARENTID	identifies a “parent” container control, such as a frame, if any
CTRLID	control identifier
EVENTMASK	list of events this control responds to
STYLE	normal (single selection) tree styles: 0 = text only 1 = text & sign 2 = text & small icon 3 = text & small icon & sign 4 = text & large icon 5 = text & large icon & sign checked (multiple selection) tree styles: 8 = text only 9 = text & sign 10 = text & small icon 11 = text & small icon & sign 12 = text & large icon 13 = text & large icon & sign hierarchical checked (multiple selection) tree styles: 16 = text only 17 = text & sign 18 = text & small icon

	19 = text & small icon & sign 20 = text & large icon 21 = text & large icon & sign
LEFT	horizontal position of the control relative to its parent form or frame
TOP	vertical position of the control relative to its parent form or frame
WIDTH	width of the control
HEIGHT	height of the control

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

GUISTATE

Node item record format:

subvalue 1	node ID (must be unique among sibling nodes)
subvalue 2	node level, starting with 1 for top-level nodes
subvalue 3	node description (text string)
subvalue 4	node icon (filename or URL)
subvalue 5	node state (0=collapsed, 1=expanded)
subvalue 6	node hint (tooltip) text (single line only)

Note: the level for each record must be either:
a) the same as the level of the previous node (node is either top-level node or a sibling of the previous node)
b) one more than the level of the previous node (node is a child of the previous node)
c) less than the previous node, but greater than zero (node is either a top-level node or a sibling of a node higher up in the tree)

Properties supported by this item:

GPFORECOLOR	foreground (text) color (default is WindowText)
GPBACKCOLOR	background color (default color is WindowBackground)
GPFONTNAME	name of default font (default is form's FONTNAME)
GPFONTSIZE	font size in points (default is form's FONTSIZE)
GPFONTBOLD	non-zero to use boldface font
GPFONTITALIC	non-zero to use italic font
GPFONTUNDERLINE	non-zero to use underlined font
GPENABLED	non-zero if control is enabled
GPVISIBLE	non-zero if control is visible
GPHELPID	help topic ID or URL
GPBORDER	0 = no border; 1 = flat border; 2 = normal border (default)
GPSTYLE	normal (single selection) tree styles: 0 = text only 1 = text & sign

	2 = text & small icon 3 = text & small icon & sign 4 = text & large icon 5 = text & large icon & sign checked (multiple selection) tree styles: 8 = text only 9 = text & sign 10 = text & small icon 11 = text & small icon & sign 12 = text & large icon 13 = text & large icon & sign hierarchical checked (multiple selection) tree styles: 16 = text only 17 = text & sign 18 = text & small icon 19 = text & small icon & sign 20 = text & large icon 21 = text & large icon & sign
GPITEMS	tree contents. Separate node records with value marks.
GPCAPTION	node caption. Use ATGUISETITEMPROP and ATGUIGETITEMPROP to access this property for an individual node.
GPICON	node icon filename or URL. Use ATGUISETITEMPROP and ATGUIGETITEMPROP to access this property for an individual node.
GPHINT	hint (or tooltip) text displayed in a balloon window when the cursor hovers over the control. Multiple lines of text are separated by subvalue marks. Use ATGUISETITEMPROP and ATGUIGETITEMPROP to access hint text (including multiline text) for an individual node.
GPSTATE	indicates if a node is collapsed (0) or expanded (1). Use ATGUISETITEMPROP and ATGUIGETITEMPROP to access this property for an individual node.
GPDRAGID	drag source ID (not necessarily the control's CTRLID). When an option group control is dropped, GUIARGS<1,2,3> contains the path to the node where the mouse button was pressed. If the mouse was not over a node, null is used.
GPDROPIDS	drop target IDs (list of drag source IDs that are allowed to be dropped on this control)
GPDEFVAL	default value of control (also sets current value).
GPVALUE	value of control (path for currently selected node).

Events supported by this item (sum the desired events to form the EVENTMASK argument):

GEVALIDATE	the user is attempting to move to another control after changing the selected node in the tree. Use this event to validate the control's new value. If validation fails, use ATGUIACTIVATE to re-activate this control. GUIARGS<1,1> = ID of the control triggering the event GUIARGS<2> = path of the selected node
GECLICK	the user selected a new node in the tree, either by clicking the node with the left mouse button, or by using the cursor keys. GUIARGS<1,1> is the selected node ID.

GECONTEXT	the user clicked the right mouse button on this control. If the mouse was over a node, the node ID is returned in GUIARGS<1,1>. Otherwise GUIARGS<1,1> is null.
GEDBLCLICK	the user double-clicked the left mouse button on this control. GUIARGS<1,1> is the selected node ID.
GESTATUS	the user has expanded or collapsed a node in the tree. GUIARGS<1,1> = node ID that was changed GUIARGS<1,2> = new node state (0 = collapsed, 1 = expanded)
GEACTIVATE	the control has become the active control. GUIARGS<1,1> is the ID of the control being deactivated.
GEDEACTIVATE	the control is no longer the active control. GUIARGS<1,1> is the ID of the control being activated.
GEDRAGDROP	the user has dropped an acceptable drag source on the control: GUIARGS<1,1> = ID of the drag source that was dropped on the control GUIARGS<1,2,1> = X coordinate within drag source where the drag began GUIARGS<1,2,2> = Y coordinate within drag source where the drag began GUIARGS<1,2,3... > = drag source details (depends on type of drag source object) GUIARGS<1,3,1> = X coordinate in control (drop target) where source was dropped GUIARGS<1,3,2> = Y coordinate in control (drop target) where source was dropped GUIARGS<1,3,3> = path of node where the mouse button was released. If the mouse was not over a node, null is used.

1.2.2.16 ATGUICREATEGAUGE

The ATGUICREATEGAUGE routine creates a *gauge* or *progress bar control*. A *gauge control* is used to display a visual representation of the percentage of some quantity or operation in progress. The *value* of a *gauge control* is a number between 0 and 100 (percent).

Calling syntax:

```
CALL ATGUICREATEGAUGE(APPID, FORMID, CTRLID, PARENTID, EVENTMASK, STYLE,
       LEFT, TOP,
       WIDTH, HEIGHT, GUIERRORS, GUISTATE)
```

Input arguments:

APPID	identifies which application control belongs to
FORMID	identifies which form control belongs to
CTRLID	control identifier
PARENTID	identifies a “parent” container control, such as a frame, if any
EVENTMASK	list of events this control responds to
STYLE	progress bar style: 0 = segmented progress bar 1 = smooth progress bar

	2 = marquee (continuously moving bar to indicate indeterminate progress)
LEFT	horizontal position of the control relative to its parent form or frame
TOP	vertical position of the control relative to its parent form or frame
WIDTH	width of the control
HEIGHT	height of the control

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

GUISTATE

Properties supported by this item:

GPFORECOLOR	foreground (text) color (default depends on Windows version)
GPBACKCOLOR	background color (default depends on Windows version)
GPFONTNAME	name of default font (default is form's FONTNAME)
GPFONTSIZE	font size in points (default is form's FONTSIZE)
GPFONTBOLD	non-zero to use boldface font
GPFONTITALIC	non-zero to use italic font
GPFONTUNDERLINE	non-zero to use underlined font
GPVISIBLE	non-zero if control is visible
GPENABLED	non-zero if control is enabled (marquee does not move if control is disabled)
GPBORDER	0 = no border (default); 1 = flat border; 2 = normal border
GPSTYLE	progress bar style: 0 = segmented progress bar 1 = smooth progress bar 2 = marquee (continuously moving bar to indicate indeterminate progress)
GPDRAGID	drag source ID (not necessarily the control's CTRLID). When a gauge control is dropped, GUIARGS<1,2,3> contains the approximate value (percent) where the mouse button was pressed.
GPDROPIDS	drop target IDs (list of drag source IDs that are allowed to be dropped on this control)
GPDEFVAL	default value of control (also sets current value)
GPVALUE	value of control (0 to 100)

Events supported by this item (sum the desired events to form the EVENTMASK argument):

GEDRAGDROP	the user has dropped an acceptable drag source on the control: GUIARGS<1,1> = ID of the drag source that was dropped on the control GUIARGS<1,2,1> = X coordinate within drag source where the drag began

	GUIARGS<1,2,2> = Y coordinate within drag source where the drag began GUIARGS<1,2,3... > = drag source details (depends on type of drag source object) GUIARGS<1,3,1> = X coordinate in control (drop target) where source was dropped GUIARGS<1,3,2> = Y coordinate in control (drop target) where source was dropped GUIARGS<1,3,3> = approximate value (percent) where the mouse button was released.

1.2.2.17 ATGUICREATEBROWSER

The ATGUICREATEBROWSER routine creates an *HTML viewer control*. The *HTML viewer* is provided to display simple HTML content, rather than provide full browser functionality. The *HTML viewer* for this version of AccuTerm GUI is implemented using a component version of Internet Explorer - future versions of AccuTerm may use a different implementation which may not have all of the capabilities of Internet Explorer.

The *HTML viewer* has a *value* property (GPVALUE) which contains the URL to display in the control, as well as a *content* (GPCONTENT) property which allows you to load HTML content directly into the viewer. The *content* property uses subvalue marks for line separators.

Like the HTML Help function (ATGUIHELP), the HTML content can contain "topic" links which will fire a Click event. You can use this event to reload the content based on the target "topic".

Calling syntax:

```
CALL ATGUICREATEBROWSER(APPID, FORMID, CTRLID, PARENTID, EVENTMASK, URL,
       LEFT, TOP,
       WIDTH, HEIGHT, GUIERRORS, GUISTATE)
```

Input arguments:

APPID	identifies which application control belongs to
FORMID	identifies which form control belongs to
CTRLID	control identifier
PARENTID	identifies a “parent” container control, such as a frame, if any
EVENTMASK	list of events this control responds to
URL	initial URL to display when control is first created or reset
LEFT	horizontal position of the control relative to its parent form or frame
TOP	vertical position of the control relative to its parent form or frame
WIDTH	width of the control
HEIGHT	height of the control

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

GUISTATE

Properties supported by this item:

GPENABLED	non-zero if control is enabled
GPVISIBLE	non-zero if control is visible
GPBORDER	0 = no border (default); 1 = flat border; 2 = normal border
GPDROPIDS	drop target IDs (list of drag source IDs that are allowed to be dropped on this control).
GPDEFVAL	default URL of viewer (also sets current value).
GPVALUE	URL to display in the viewer.
GPCONTENT	HTML content to display in the viewer. Use subvalue marks to separate multiple lines.

Events supported by this item (sum the desired events to form the EVENTMASK argument):

GECLICK	the user clicked a "topic" link. GUIARGS<1,1> contains the link target.
GEDRAGDROP	the user has dropped an acceptable drag source on the control: GUIARGS<1,1> = ID of the drag source that was dropped on the control GUIARGS<1,2,1> = X coordinate within drag source where the drag began GUIARGS<1,2,2> = Y coordinate within drag source where the drag began GUIARGS<1,2,3... > = drag source details (depends on type of drag source object) GUIARGS<1,3,1> = X coordinate in control (drop target) where source was dropped GUIARGS<1,3,2> = Y coordinate in control (drop target) where source was dropped

1.2.2.18 ATGUICREATEMENU

The ATGUICREATEMENU routine creates a *menu* for a form or MDI application. A *menu* may be a top-level *menu* or a pop-up *menu.* A *menu item* is used to trigger an action when the user “clicks” a menu item or presses the item’s associated shortcut key. To detect the “click”, the event mask must contain GECLICK. The argument returned in the click event contains the *menu item* ID. The *menu* does not have a *value*.

Menus normally do not trigger *validate* events. Often, menu selections are used to perform utility functions like printing or displaying help. However, for cases where a menu action should cause validation, an option is provided to trigger the *validate* event of the active control when the item is clicked.

Menus are represented using a nested structure. Top-level menu items are at level 1. The ubiquitous "File", "Edit", "View"... menu bar at the top of most Windows are examples of top-level menu items. Clicking a top-level menu item typically opens a sub-menu, shown as a drop-down menu under the top-level menu item. Items in the sub-menu are level 2. These sub-menus may themselves open other sub-

menus at level 3, etc. When constructing the menu structure, the first item must be at level 1, followed by all level 2 items under the level 1 item. Level 3 items may follow the level 2 item they are under, etc.

When a menu is created for an MDI application, the menu is visible when the active child form does not have a menu, or when there are no open child forms. When a child form with its own menu is active, the child's menu replaces the MDI menu.

Calling syntax:

```
CALL ATGUICREATEMENU(APPID, FORMID, MENUID, STYLE, ITEMS, GUIERRORS,
      GUISTATE)
```

Input arguments:

APPID	identifies which application control belongs to
FORMID	identifies which form control belongs to (null if menu belongs to the MDI application)
MENUID	menu identifier
STYLE	0 = top-level menu, 1 = pop-up menu
ITEMS	menu item records separated by value marks - each menu item record contains 8 fields separated by subvalue marks.

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

Menu item record format:

subvalue 1	menu item ID (unique within menu). Use a "-" to indicate a separator between menu items. Special item IDs may be used for certain standard operations: *CUT, *COPY, *PASTE, *SELECTALL, *PRINT, *PRINTSETUP, *CLOSE (MDI child form), *EXIT, *TILE, *CASCADE, *WINDOW (MDI child window list), *HELP, *ABOUT (these special IDs begin with an asterisk).
subvalue 2	menu item nesting level (must begin with level 1). Following is an example showing the nesting using indentation, with the level in parentheses: File (1) Open (2) New (2) Document (3) Folder (3) Close (2) Exit (2)
subvalue 3	menu item caption

subvalue 4	optional menu item icon filename or URL
subvalue 5	non-zero if item is enabled
subvalue 6	non-zero if item is visible
subvalue 7	shortcut key code (see KEY… constants in ATGUIEQUATES)
subvalue 8	non-zero if item causes Validate event to fire when clicked

Properties supported by this item:

GPENABLED	use ATGUIGETITEMPROP and ATGUISETITEMPROP to access individual menu item enable state.
GPVISIBLE	non-zero if menu is visible. Use ATGUIGETITEMPROP and ATGUISETITEMPROP to access individual menu item visible state.
GPCAPTION	use ATGUIGETITEMPROP and ATGUISETITEMPROP to access individual menu item caption.
GPICON	use ATGUIGETITEMPROP and ATGUISETITEMPROP to access individual menu item icon filename or URL.
GPITEMS	menu item records separated by value marks - each menu item record contains 8 fields separated by subvalue marks (see "Menu item record format" above). An individual item can be accessed by ATGUIGETPROP and ATGUISETPROP by specifying non-zero ROW argument, or by ATGUIGETITEMPROP and ATGUISETITEMPROP by specifying the menu item ID.

Events supported by this item (sum the desired events to form the EVENTMASK argument):

GECLICK	the user clicked the left mouse button on a menu item. GUIARGS<1,1> is the ID of the menu item clicked.

1.2.2.19 ATGUICREATETOOLBAR

The ATGUICREATETOOLBAR routine creates a *toolbar* for a form or MDI application. A *toolbar* is used to trigger an action when the user “clicks” a button in the *toolbar*. To detect the “click”, the event mask must contain GECLICK. The argument returned in the click event contains the *tool item* ID. The *toolbar* does not have a *value*.

Toolbars normally do not trigger *validate* events. Often, toolbar buttons are used to perform utility functions like printing or displaying help. However, for cases where a button action should cause validation, an option is provided to trigger the *validate* event of the active control when the button is clicked.

When a toolbar is created on an MDI application, the toolbar is visible when the active child form does not have a menu or toolbar, or when there are no open child forms. When a child form with its own menu and toolbar is active, the child’s toolbar replaces the MDI toolbar.

Calling syntax:

```
CALL ATGUICREATETOOLBAR(APPID, FORMID, CTRLID, POSITION, STYLE, ITEMS,
      GUIERRORS, GUISTATE)
```

Input arguments:

APPID	identifies which application control belongs to
FORMID	identifies which form control belongs to (null if toolbar belongs to an MDI application)
CTRLID	toolbar identifier
POSITION	0 = floating, 1 = top, 2 = bottom, 3 = left, 4 = right
STYLE	0 = small icons, 1 = large icons, 2 = small icon with caption, 3 = large icon with caption
ITEMS	toolbar item records separated by value marks. Each toolbar item record contains 9 fields separated by subvalue marks.

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

GUISTATE

Toolbar item record format:

subvalue 1	button item ID (unique within toolbar). Use a "-" to indicate a separator between buttons. Special item IDs may be used for certain standard operations: *CUT, *COPY, *PASTE, *SELECTALL, *PRINT, *PRINTSETUP, *CLOSE (MDI child form), *EXIT, *TILE, *CASCADE, *HELP, *ABOUT (these special IDs begin with an asterisk).
subvalue 2	reserved - use null
subvalue 3	button caption (displayed to right of icon); requires toolbar style 2 or 3 (see below).
subvalue 4	button icon filename or URL
subvalue 5	non-zero if item is enabled
subvalue 6	non-zero if item is visible
subvalue 7	reserved - use null
subvalue 8	non-zero if button causes Validate event to fire when clicked
subvalue 9	tooltip help text

Properties supported by this item:

GPENABLED	use ATGUIGETITEMPROP and ATGUISETITEMPROP to access individual toolbar button enabled state.
GPVISIBLE	non-zero if toolbar is visible. Use ATGUIGETITEMPROP and ATGUISETITEMPROP to access individual toolbar button visible state.
GPALIGN	toolbar position: 0 = floating, 1 = top, 2 = bottom, 3 = left, 4 = right.
GPSTYLE	0 = small icons / no caption, 1 = large icons / no caption, 2 = small icons / show captions, 3 = large icons / show captions.
GPICON	use ATGUIGETITEMPROP and ATGUISETITEMPROP to access individual toolbar button icon filename or URL.

GPHINT	use ATGUIGETITEMPROP and ATGUISETITEMPROP to access individual toolbar button hint (or tooltip) text displayed in a balloon window when the cursor hovers over a button.
GPITEMS	toolbar item records separated by value marks. Each toolbar item record contains 9 fields separated by subvalue marks (see "Toolbar item record format" above). An individual item can be accessed by ATGUIGETPROP and ATGUISETPROP by specifying non-zero ROW argument, or by ATGUIGETITEMPROP and ATGUISETITEMPROP by specifying the button ID.

Events supported by this item (sum the desired events to form the EVENTMASK argument):

GECLICK	the user clicked the left mouse button on a toolbar button. GUIARGS<1,1> is the ID of the toolbar button clicked.

1.2.2.20 ATGUICREATESTATUSBAR

The ATGUICREATESTATUSBAR routine creates a *status bar* for a form or MDI application. A *status bar* consists of a series of panels which display application state information. It is normally displayed at the bottom of a form. A *status bar* does not have a *value*.

When a status bar is created on an MDI application, the status bar is visible when the active child form does not have a menu toolbar or status bar, or when there are no open child forms. When a child form with its own menu, toolbar and status bar is active, the child's status bar replaces the MDI status bar.

Calling syntax:

```
CALL ATGUICREATEstatus bar(APPID, FORMID, CTRLID, STYLE, ITEMS,
       GUIERRORS, GUISTATE)
```

Input arguments:

APPID	identifies which application control belongs to
FORMID	identifies which form control belongs to (null if status bar belongs to an MDI application)
CTRLID	status bar identifier
STYLE	0 = small icons, 1 = large icons
ITEMS	status bar item records separated by value marks. Each status bar item record contains 9 fields separated by subvalue marks.

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

status bar item record format:

subvalue 1	panel item ID (unique within status bar)
subvalue 2	reserved - use null
subvalue 3	panel caption text
subvalue 4	optional panel icon filename or URL
subvalue 5	non-zero if item is enabled
subvalue 6	non-zero if item is visible
subvalue 7	reserved - use null
subvalue 8	reserved - use null
subvalue 9	tooltip help text
subvalue 10	panel size: 0 = auto size, -1 = spring (fills remaining unused space), else actual size
subvalue 11	panel alignment: 0 = left, 1 = right, 2 = center

Note 1: when a panel size is set to -1 (spring), the panel size is adjusted to fill up the remaining space on the status bar. If multiple panels are set to "spring", the extra space is divided among them.

Note 2: according to Windows Dev Center, status bar tooltip text is only displayed in two situations:

- when the panel contains only an icon
- when the panel size causes the panel text to be truncated

Properties supported by this item:

GPENABLED	use ATGUIGETITEMPROP and ATGUISETITEMPROP to access individual status bar panel enabled state.
GPVISIBLE	non-zero if status bar is visible. Use ATGUIGETITEMPROP and ATGUISETITEMPROP to access individual status bar panel visible state.
GPCAPTION	use ATGUIGETITEMPROP and ATGUISETITEMPROP to access individual status bar panel caption.
GPALIGN	status bar position: 0 = floating, 1 = top, 2 = bottom, 3 = left, 4 = right. Use ATGUIGETITEMPROP and ATGUISETITEMPROP to access individual status bar panel alignment.
GPWIDTH	use ATGUIGETITEMPROP and ATGUISETITEMPROP to access individual status bar panel size.
GPSTYLE	0 = small icons, 1 = large icons
GPICON	use ATGUIGETITEMPROP and ATGUISETITEMPROP to access individual status bar panel icon filename or URL.
GPHINT	use ATGUIGETITEMPROP and ATGUISETITEMPROP to access individual status bar panel hint (or tooltip) text displayed in a balloon window when the cursor hovers over a panel.
GPITEMS	status bar item records separated by value marks. Each status bar item record contains 11 fields separated by subvalue marks (see "Status bar item record format" above). An individual item can be accessed by ATGUIGETPROP and ATGUISETPROP by specifying non-zero ROW argument, or by ATGUIGETITEMPROP and ATGUISETITEMPROP by specifying the panel ID.

Events supported by this item (sum the desired events to form the EVENTMASK argument):

GECLICK	the user clicked the left mouse button on a status bar panel. GUIARGS<1,1> is the ID of the panel clicked.
GEDBLCLICK	the user double-clicked the left mouse button on a status bar panel. GUIARGS<1,1> is the ID of the panel clicked.
GECONTEXT	the user clicked the right mouse button on a status bar panel. GUIARGS<1,1> is the ID of the panel clicked.

1.2.2.21 ATGUICREATETIMER

The ATGUICREATETIMER routine creates a timer control that fires a Timer (GETIMER) event after a specified interval. The timer may be one-shot or repeating. For one-shot timers, you can restart the timer by setting the Timeout (GPTIMEOUT) property to the desired timeout in milliseconds. For repeating timers, setting Enabled (GPENABLED) to False (0) will disable the timer.The Timeout (GPTIMEOUT) value can be changed at any time.

Using the timer control may be more convenient than calling ATGUICHECKEVENT with a timeout because the interval can be controlled more precisely.

Multiple timers are supported.

Calling syntax:

```
CALL ATGUICREATETIMER(APPID, FORMID, CTRLID, STYLE, TIMEOUT, GUIERRORS,
      GUISTATE)
```

Input arguments:

APPID	identifies which application control belongs to
FORMID	identifies which form control belongs to
CTRLID	control identifier
PARENTID	identifies a “parent” container control, such as a frame, if any
STYLE	0 for one-shot timer, 1 for repeating timer
TIMEOUT	timeout interval in milliseconds

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

Properties supported by this item:

GPENABLED	non-zero if control is enabled
GPSTYLE	0 for one-shot timer, 1 for repeating timer
GPTIMEOUT	timeout interval in milliseconds

Events supported by this item (sum the desired events to form the EVENTMASK argument):

GETIMER	the timeout interval has expired

1.2.2.22 ATGUIDELETE

The ATGUIDELETE function is called to delete (close) an application, form or control.

Calling syntax:

```
CALL ATGUIDELETE(APPID, FORMID, CTRLID, GUIERRORS, GUISTATE)
```

Input arguments:

APPID	application identifier
FORMID	form identifier (null if deleting entire application)
CTRLID	control (or menu) identifier (null if deleting entire form or application)

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.3 Global Objects

The AccuTerm GUI runtime has several Global objects: the Root, Printer and Screen. You can use ATGUIGETPROP and ATGUISETPROP to access the properties of these global objects.

The Root object
The Root object is the common parent shared by all *application* objects. The Root object's AppID is GXROOT; pass NULL for the FormID and CtrlID arguments when you call ATGUIGETPROP or ATGUISETPROP. The Root object supports only the GPSTATUS property, which can be used to change the message text in the "This session is running a GUI application" status panel which is displayed on the terminal screen while a GUI application is running.

Properties supported by the **Root object**:

GPSTATUS	returns status information about the GUI runtime environment: COL = -1 : returns (or sets) the GUI status message text displayed in the terminal screen status box COL = 0 : returns the number of visible forms of all application objects COL = 1 : returns the ID of the currently active control COL = 2 : returns a multi-valued list of application objects:

	ROW = 0 : return all application objects ROW = 1 : return only MDI application objects ROW = 2 : return only SDI application objects

The Screen object
The Screen object provides access to information about the display resolution of the user's workstation. You can retrieve both the actual and useable size of the display, in any supported scale mode. The Screen object's AppID is GXSCREEN; pass NULL for the FormID and CtrlID arguments when you call ATGUIGETPROP or ATGUISETPROP.

Properties supported by the **Screen object**:

GPWIDTH	returns the display width. Pass the desired scale mode in the ROW argument of ATGUIGETPROP (see the Application object for a list of supported scale modes). To return the full width, pass zero in the COL argument; to return the useable width, pass 1 in COL.
GPHEIGHT	returns the display height. Pass the desired scale mode in the ROW argument of ATGUIGETPROP (see the Application object for a list of supported scale modes). To return the full height, pass zero in the COL argument; to return the useable height, pass 1 in COL.

The Printer object
The Printer object provides access to information about the currently selected printer on the user's workstation. You can retrieve both the actual and useable size of the display, in any supported scale mode. The Printer object's AppID is GXPRINTER; pass NULL for the FormID and CtrlID arguments when you call ATGUIGETPROP or ATGUISETPROP.

Properties supported by the **Printer object**:

GPWIDTH	returns the page width. Pass the desired scale mode in the ROW argument of ATGUIGETPROP (see the application object for a list of supported scale modes). To return the physical page width, pass zero in the COL argument; to return the useable page width, pass 1 in COL.
GPHEIGHT	returns the page height. Pass the desired scale mode in the ROW argument of ATGUIGETPROP (see the application object for a list of supported scale modes). To return the physical page height, pass zero in the COL argument; to return the useable page height, pass 1 in COL.
GPPRINTERNAME	sets or retrieves the current printer name.
GPORIENTATION	sets or retrieves the page orientation (1 = portrait, 2 = landscape).
GPPAPERSOURCE	sets or retrieves the paper bin for the current printer. The actual value is dependent on the printer driver, so it is best to retrieve this value after calling ATGUIPRINT2 and use this property to restore the retrieved value.
GPPAPERSIZE	sets or retrieves the paper size for the current printer. Standard paper sizes are: 1 = letter, 5 = legal, 8 = A3, 9 = A4. Other sizes are dependent on the printer driver.
GPPRINTQUALITY	sets or retrieves the print quality. This value may be in dots per inch or one of the following: -1 for draft, -2 for low quality, -3 for medium quality, -4 for high quality.
GPPRINTCOPIES	sets or retrieves the number of copies for the current printer. Note: not all printers support multiple copies.

GPPRINTDUPLEXMODE	sets or retrieves the duplex mode for the current printer: 1 = single sided printing, 2 = double-sided horizontal, 3 = double sided vertical.
GPPRINTCOLORMODE	sets or retrieves the color mode for the current printer: 1 = monochrome, 2 = color.

1.2.4 GUI State Functions

1.2.4.1 ATGUIACTIVATE

The ATGUIACTIVATE function is called activate an application, form or control. Use this function to restore activation to a control which failed validation in the GEVALIDATE event.

Calling syntax:

```
CALL ATGUIACTIVATE(APPID, FORMID, CTRLID, GUIERRORS, GUISTATE)
```

Input arguments:

APPID	application identifier
FORMID	form identifier (null if activating an application)
CTRLID	control (or menu) identifier (null if activating a form)

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.4.2 ATGUISHOW

The ATGUISHOW routine makes a form, control or menu item visible.

Calling syntax:

```
CALL ATGUISHOW(APPID, FORMID, CTLID, MENUID, GUIERRORS, GUISTATE)
```

Input arguments:

APPID	application identifier
FORMID	form identifier
CTLID	control identifier
MENUID	menu item identifier (only if CTLID refers to a menu, popup or toolbar, otherwise null)

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

Note: contrary to normal design principles, showing a Form may trigger Activate and Deactivate events for the form that becomes active or inactive. There may also be Activate, Deactivate and Validate events triggered for the control being activated or deactivated when the active form changes.

1.2.4.3 ATGUIHIDE

The ATGUIHIDE routine makes a form, control or menu item invisible.

Calling syntax:

```
CALL ATGUIHIDE(APPID, FORMID, CTLID, MENUID, GUIERRORS, GUISTATE)
```

Input arguments:

APPID	application identifier
FORMID	form identifier
CTLID	control identifier
MENUID	menu item identifier (only if CTLID refers to a menu, popup or toolbar, otherwise null)

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

Note: contrary to normal design principles, hiding a Form may trigger Activate and Deactivate events for the form that becomes active or inactive. There may also be Activate, Deactivate and Validate events triggered for the control being activated or deactivated when the active form changes.

1.2.4.4 ATGUIENABLE

The ATGUIENABLE routine enables a form, control or menu item.

Calling syntax:

```
CALL ATGUIENABLE(APPID, FORMID, CTLID, MENUID, GUIERRORS, GUISTATE)
```

Input arguments:

APPID	application identifier
FORMID	form identifier
CTLID	control identifier
MENUID	menu item identifier if CTLID refers to a menu, popup or toolbar, option index if CTLID refers to an option group; otherwise NULL.

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.4.5 ATGUIDISABLE

The ATGUIDISABLE routine disables a form, control or menu item.

Calling syntax:

```
CALL ATGUIDISABLE(APPID, FORMID, CTLID, MENUID, GUIERRORS, GUISTATE)
```

Input arguments:

APPID	application identifier
FORMID	form identifier
CTLID	control identifier
MENUID	menu item identifier if CTLID refers to a menu, popup or toolbar, option index if CTLID refers to an option group; otherwise null

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.4.6 ATGUIGETACTIVE

The ATGUIGETACTIVE routine returns the active application, form and control ID. This routine may be useful when handling menu click events, where the appropriate action may be tailored to suit the currently active control. If the active control cannot be identified, null is returned.

Calling syntax:

```
CALL ATGUIGETACTIVE(APPID,FORMID, CTLID, GUIERRORS, GUISTATE)
```

Input arguments:

none

Output arguments:

APPID	active application identifier
FORMID	active form identifier
CTRLID	active control identifier
GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.4.7 ATGUIMOVE

The ATGUIMOVE routine is called to reposition, and optionally to resize a form or control.

Calling syntax:

```
CALL ATGUIMOVE(APPID, FORMID, CTRLID, LEFT, TOP, WIDTH, HEIGHT,
       GUIERRORS, GUISTATE)
```

Input arguments:

APPID	application identifier
FORMID	form identifier
CTRLID	control identifier (null if moving form)
LEFT	new horizontal position of form or control (null to keep current left)
TOP	new vertical position of form or control (null to keep current top)
WIDTH	new width of form or control (null to keep current width)
HEIGHT	new height of form or control (null to keep current height)

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.4.8 ATGUISORT

The ATGUISORT routine sorts the items in a list, combo or grid control.

Calling syntax:

```
CALL ATGUISORT(APPID, FORMID, CTLID, COLS, GUIERRORS, GUISTATE)
```

Input arguments:

APPID	application identifier
FORMID	form identifier
CTLID	control identifier

COLS	columns to sort. The column numbers are separated by value marks. Each column may have an optional sort order (A or D) in the 2nd subvalue. For example, to sort column 2 in ascending order, and column 5 in descending order:
	`COLS = ''`
	`COLS<1,1,1> = 2`
	`COLS<1,1,2> = 'A'`
	`COLS<1,2,1> = 5`
	`COLS<1,2,2> = 'D'`
	Note: the order that the columns are specified in the COLS *argument does not affect the sort order. The column precedence is always left to right.*

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

The sort for each control type (list, combo, grid) may not be identical, especially when duplicate values exist in one of the sorted columns.

The ColDataType (GPCOLDATATYPE) property can be used to specify the data type (and indirectly the sort type) for columns in list, combo or grid controls. If a column contains date or time data, it is necessary to set the data type for the column to GDDATE or GDTIME. Columns with unspecified data type are sorted as alpha or numeric, depending on the column alignment. Right-aligned columns assume numeric values. Any digits that precede non-numeric characters are treated as a numeric value and the rest of the data is ignored.

Be careful when referencing list items by index or row after calling ATGUISORT as the order of the items will be different!

1.2.5 GUI Property Functions

1.2.5.1 ATGUISETPROP

The ATGUISETPROP routine sets the value of a property of a GUI application, form or control.

Calling syntax:

```
CALL ATGUISETPROP(APPID, FORMID, CTRLID, PROPERTY, COL, ROW,_
      VALUE, GUIERRORS, GUISTATE)
```

Input arguments:

APPID	application identifier
FORMID	form identifier (null if setting application property)
CTRLID	control identifier (null if setting form or application property)
PROPERTY	property code (see each control or form topic for specific properties; property codes begin with GP... , eg. GPFORECOLOR)
COL	if property accepts multiple columns, specifies column number; zero indicates all columns
ROW	if property accepts multiple rows, specifies row number; zero indicates all rows
VALUE	property value. Some properties, such as lists, accept multiple values - separate multiple rows with value marks, separate multiple columns (within multiple rows) with subvalue marks.

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.5.2 ATGUIGETPROP

The ATGUIGETPROP routine retrieves the value of a property of a GUI application, form or control.

Calling syntax:

```
CALL ATGUIGETPROP(APPID, FORMID, CTRLID, PROPERTY, COL, ROW,
      VALUE, GUIERRORS, GUISTATE)
```

Input arguments:

APPID	application identifier
FORMID	form identifier (null if setting application property)
CTRLID	control identifier (null if setting form or application property)
PROPERTY	property code (see each control or form topic for specific properties;

	property codes begin with GP... without the dot, eg. GPFORECOLOR)
COL	if property accepts multiple columns, specifies column number; zero indicates all columns
ROW	if property accepts multiple rows, specifies row number; zero indicates all rows

Output arguments:

VALUE	property value. Some properties, such as lists, contain multiple values - multiple rows are separated by value marks, multiple columns (within multiple rows) are separated by subvalue marks.
GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.5.3 ATGUISETPROPS

The ATGUISETPROPS routine sets one or more properties of one or more controls. This function is like calling ATGUISETPROP(..., ..., ...) multiple times.

Calling syntax:

```
CALL ATGUISETPROPS(APPID, FORMID, CTRLIDS, PROPS, VALUES, GUIERRORS,
      GUISTATE)
```

Input arguments:

APPID	application identifier
FORMID	form identifier
CTRLIDS	control identifiers separated by attribute marks
PROPS	property codes separated by attribute marks
VALUES	corresponding property values separated by attribute marks; multiple rows within a value are separated by value marks and multiple columns within a row are separated by subvalue marks.

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.5.4 ATGUIGETPROPS

The ATGUIGETPROPS routine retrieves the value of a property of a GUI application, form or control.

Calling syntax:

```
CALL ATGUIGETPROPS(APPID, FORMID, CTRLIDS, PROPS, VALUES, GUIERRORS,
      GUISTATE)
```

Input arguments:

APPID	application identifier
FORMID	form identifier
CTRLIDS	control identifiers separated by attribute marks
PROPS	corresponding property codes separated by attribute marks

Output arguments:

VALUES	corresponding property values separated by attribute marks. Multiple rows within a value are separated by value marks, multiple columns within a row are separated by subvalue marks.
GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.5.5 ATGUISETITEMPROP

The ATGUISETITEMPROP routine sets the value of a property of a specific item of a control. This routine provides access to individual elements of the Items property of Tree, Menu, Toolbar and Status bar controls. See the individual object description for information on the properties supported by each object type.

Calling syntax:

```
CALL ATGUISETITEMPROP(APPID, FORMID, CTRLID, ITEMID, PROPERTY, COL, ROW,
      VALUE, GUIERRORS, GUISTATE)
```

Input arguments:

APPID	application identifier
FORMID	form identifier
CTRLID	control identifier
ITEMID	ID of menu, toolbar or tree item
PROPERTY	property code (see each control topic for specific properties)
COL	reserved - use zero
ROW	reserved - use zero
VALUE	property value

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.5.6 ATGUIGETITEMPROP

The ATGUIGETITEMPROP routine retrieves the value of a property of a specific item of a control. This routine provides access to individual elements of the Items property of Tree, Menu, Toolbar and Status bar controls. See the individual object description for information on the properties supported by each object type.

Calling syntax:

```
CALL ATGUIGETITEMPROP(APPID, FORMID, CTRLID, ITMIT, PROPERTY, COL, ROW,
      VALUE, GUIERRORS, GUISTATE)
```

Input arguments:

APPID	application identifier
FORMID	form identifier
CTRLID	control identifier
ITEMID	ID of menu, toolbar or tree item
PROPERTY	property code (see each control topic for specific properties)
COL	reserved - use zero
ROW	reserved - use zero

Output arguments:

VALUE	property value
GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.5.7 ATGUILOADVALUES

The ATGUILOADVALUES routine is called to load a set of values onto a GUI form.

Calling syntax:

```
CALL ATGUILOADVALUES(APPID, FORMID, CTRLIDS, VALUES, GUIERRORS,
      GUISTATE)
```

Input arguments:

APPID	application identifier
FORMID	form identifier
CTRLIDS	control identifiers separated by attribute marks.
VALUES	corresponding values separated by attribute marks. Multiple rows within a value are separated by value marks and multiple columns within a row are separated by subvalue marks

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.5.8 ATGUIGETVALUES

The ATGUIGETVALUES routine is called to retrieve a set of values from a GUI form.

Calling syntax:

```
CALL ATGUIGETVALUES(APPID, FORMID, CTRLIDS, VALUES, GUIERRORS, GUISTATE)
```

Input arguments:

APPID	application identifier
FORMID	form identifier
CTRLIDS	control identifiers separated by attribute marks.

Output arguments:

VALUES	corresponding values separated by attribute marks. Multiple rows within a value are separated by value marks and multiple columns within a row are separated by subvalue marks
GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.5.9 ATGUIGETUPDATES

The ATGUIGETUPDATES routine is called to retrieve all updated values from a GUI form.

Calling syntax:

```
CALL ATGUIGETUPDATES(APPID, FORMID, CTRLIDS, VALUES, GUIERRORS,
      GUISTATE)
```

Input arguments:

APPID	application identifier
FORMID	form identifier

Output arguments:

CTRLIDS	control identifiers whose updated values have been retrieved; IDs are separated with attribute marks.
VALUES	corresponding values separated by attribute marks. Multiple rows within a value are separated by value marks and multiple columns within a row are separated by subvalue marks.
GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.5.10 ATGUIRESET

The ATGUIRESET routine is called to reset the *value* of all controls on a form to their *default value*. The *default value* for a control is the *value* the control was assigned when it was first created.

Note: ATGUIRESET does not reset any other properties, such as the Items property of a list control, or the enabled or visible state of any control.

Calling syntax:

```
CALL ATGUIRESET(APPID, FORMID, GUIERRORS, GUISTATE)
```

Input arguments:

APPID	application identifier
FORMID	form identifier

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.5.11 ATGUICLEAR

The ATGUICLEAR routine clears the value of any control. For list or combo controls, the list is also cleared.

Calling syntax:

```
CALL ATGUICLEAR(APPID, FORMID, CTLID, GUIERRORS, GUISTATE)
```

Input arguments:

APPID	application identifier
FORMID	form identifier
CTLID	control identifier (listbox, combo or grid control)

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.5.12 ATGUIINSERT

The ATGUIINSERT routine inserts a new row into a *grid* control. The row to be inserted before is specified. Specify zero to add a new row at the end of the *grid*.

Calling syntax:

```
CALL ATGUIINSERT(APPID, FORMID, CTLID, ROW, GUIERRORS, GUISTATE)
```

Input arguments:

APPID	application identifier
FORMID	form identifier
CTLID	control identifier (grid control)
ROW	row to insert before; zero to add new row at end

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.5.13 ATGUIINSERTITEMS

The ATGUIINSERTITEMS routine inserts one or more new items (rows) into a *grid, list* or *combo-box* control. The row to be inserted before is specified. Specify zero to add the new items after the last item.

Calling syntax:

```
CALL ATGUIINSERTITEMS(APPID, FORMID, CTLID, ROW, ITEMS, GUIERRORS,
      GUISTATE)
```

Input arguments:

APPID	application identifier
FORMID	form identifier
CTLID	control identifier (grid control)
ROW	row to insert before; zero to add new items at end
ITEMS	items to insert (same format as GPVALUE for grid, same format as GPITEMS for list or combo-box).

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.5.14 ATGUIREMOVE

The ATGUIREMOVE routine deletes a row from a *grid* control. The row to be deleted is specified.

Calling syntax:

```
CALL ATGUIREMOVE(APPID, FORMID, CTLID, ROW, GUIERRORS, GUISTATE)
```

Input arguments:

APPID	application identifier
FORMID	form identifier
CTLID	control identifier (grid control)
ROW	row to delete

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.5.15 ATGUIREMOVEITEMS

The ATGUIREMOVEITEMS routine deletes one or more items (rows) from a *grid, list* or *combo-box* control. The first item (row) to be deleted and number of items (rows) to delete are specified.

Calling syntax:

```
CALL ATGUIREMOVEITEMS(APPID, FORMID, CTLID, ROW, COUNT, GUIERRORS,
      GUISTATE)
```

Input arguments:

APPID	application identifier
FORMID	form identifier
CTLID	control identifier (grid control)
ROW	first row to delete
COUNT	number of rows to delete

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.5.16 ATGUICUT

The ATGUICUT routine copies the selected value of a control to the clipboard, then deletes the selection. If there is no selection, no operation is done. This method is valid for edit, combo and grid controls.

Calling syntax:

```
CALL ATGUICUT(APPID, FORMID, CTLID, GUIERRORS, GUISTATE)
```

Input arguments:

APPID	application identifier
FORMID	form identifier
CTLID	control identifier (edit, combo or grid control)

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.5.17 ATGUICOPY

The ATGUICOPY routine copies the selected value of a control to the clipboard. If there is no selection, no operation is done. This method is valid for edit, combo, list, grid and tree controls.

Calling syntax:

```
CALL ATGUICOPY(APPID, FORMID, CTLID, GUIERRORS, GUISTATE)
```

Input arguments:

APPID	application identifier
FORMID	form identifier
CTLID	control identifier (edit, combo, list, grid or tree control)

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.5.18 ATGUIPASTE

The ATGUIPASTE routine copies the text contents of the clipboard to a control. If the control has a current selection, the selection is replaced by the clipboard contents. Otherwise, the clipboard contents are inserted at the current location. This method is valid for edit, combo and grid controls.

Calling syntax:

```
CALL ATGUIPASTE(APPID, FORMID, CTLID, GUIERRORS, GUISTATE)
```

Input arguments:

APPID	application identifier
FORMID	form identifier
CTLID	control identifier (edit, combo or grid control)

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

GUISTATE

1.2.6 Standard Properties

All controls (and forms and the MDI application) have certain standard properties. These properties can be retrieved and changed for all controls, forms and the MDI application. The standard properties are:

GPEVENTMASK	a "mask" of all events that the control is to trigger; add all GE … constants together to form the event mask.
GPLEFT	horizontal position. Forms are relative to the screen, controls are relative to their container control or form.
GPTOP	vertical position. Forms are relative to the screen, controls are relative to their container control. or form
GPWIDTH	width of the form or control
GPHEIGHT	height of the form or control
GPTABSTOP	non-zero if control is in the "tab order" for the form. Set to zero to remove control from the "tab order".
GPFORECOLOR GPBACKCOLOR	for objects which support color properties, the color value is computed using Red, Green and Blue components using the following formula: RGB = Red + (Green * 256) + (Blue * 65536) The range of each color component is 0 to 255. The RGB values for some common colors are defined in ATGUIEQUATES. Several system colors (window background, desktop, highlight, etc.) are also defined in ATGUIEQUATES.
GPDATATYPE	specifies one of the enumerated data types. This property only applies to edit, combo box and grid controls: GDANY: no type checking GDALPHA: alpha characters GDALPHANUM: alpha or numeric characters GDBOOL: boolean (yes/no/true/false) GDCURRENCY: monetary value GDDATE: date value GDFINANCIAL: numeric value (up to 4 decimal places) GDNUMERIC: numeric value (digits 0-9 with optional minus sign) GDPERCENT: percentage GDPHONE: phone number GDSSN: social security number GDTIME: time value GDZIPCODE: 5 or 9 digit zip code
GPCHANGED	zero if control's value has not been updated by the user,

	otherwise 1. The Changed property of a container, such as a form, reflects the Changed property of any contained controls - that is if any control's Changed property is set, the container's Changed property is also set. You can test the Changed property of a form to determine if any controls within the form have been updated. If you modify the Changed property of a control using ATGUISETPROP, the Changed property of the control's parent objects (form, frame, tab, etc.) is adjusted.
GPHINT	hint (or tooltip) text displayed in a balloon window when the cursor hovers over a control. Multiple lines of text are separated by subvalue marks.
GPEXTENSION	user-defined text string which can be associated with any GUI object. The string may not contain attribute marks, but value and subvalue marks are permitted.

1.2.7 GUI Event Processing Functions

1.2.7.1 ATGUIWAITEVENT

The AccuTerm GUI environment is an *event driven* environment. In this environment, the host program creates GUI interface items using the ATGUICREATE… routines, then enters an *event loop*. The *event loop* processes user input in the form of *events*. Most of the GUI interface items which can be used in an AccuTerm GUI project are capable of generating various *events*, such as click, double-click, close, validate, etc. There is no enforced order for user input, and each *event* is identified with a unique identifier (App ID, Form ID, Control ID) as well as an *event code* which identifies the particular *event. Events* are processed sequentially.

Event processing is handled by the ATGUIWAITEVENT routine. The ATGUIWAITEVENT routine is called at the top of your *event loop.* This event processing continues until the Quit event (GEQUIT) is received. Once the Quit event has been received, the GUI application has been shutdown.

Calling syntax:

```
CALL ATGUIWAITEVENT(APPID, FORMID, CTLID, EVENT, GUIARGS, GUIERRORS,
      GUISTATE)
```

Input arguments:

none

Output arguments:

APPID	application identifier
FORMID	form identifier
CTRLID	control (or menu) identifier which triggered the event
EVENT	event code (see EVENT documentation for each control type)
GUIARGS	any event argument values separated by value marks (see EVENT documentation for each control type)
GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.7.2 ATGUICHECKEVENT

This routine is similar to ATGUIWAITEVENT, but includes a timeout argument. If an event occurs before the timeout expires, the event is returned, otherwise a zero is returned in the EVENT argument. A timeout of 0 (zero) may be specified to check for any un-processed events. The *event* is identified with a unique identifier (App ID, Form ID, Control ID) and an *event code* which identifies the particular *event. Events* are processed sequentially.

Calling syntax:

```
CALL ATGUICHECKEVENT(TIMEOUT, APPID, FORMID, CTLID, EVENT, GUIARGS,
       GUIERRORS, GUISTATE)
```

Input arguments:

TIMEOUT	timeout value in milliseconds

Output arguments:

APPID	application identifier
FORMID	form identifier
CTRLID	control (or menu) identifier which triggered the event
EVENT	event code (see EVENT documentation for each control type)
GUIARGS	any event argument values separated by value marks (see EVENT documentation for each control type)
GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.7.3 ATGUIPOSTEVENT

ATGUIPOSTEVENT is used to simulate events. Simulated events may be posted to the before or after any pending events. ATGUIPOSTEVENT is most useful to communicate between different applications running together under control of a main program. For example, if a menu item on a Customer application should open a Shipper application, the customer application can post a “LOAD” pseudo-event for the Shipper application. The multiple application model will dispatch the event to the correct application subroutine to handle the event, which will then load the requested application.

Calling syntax:

```
CALL ATGUIPOSTEVENT(APPID, FORMID, CTLID, EVENT, GUIARGS, POSITION,
      GUIERRORS, GUISTATE)
```

Input arguments:

APPID	application identifier
FORMID	form identifier
CTRLID	control (or menu) identifier which triggered the event
EVENT	event code (see EVENT documentation for each control type)
GUIARGS	any event argument values separated by value marks (see EVENT documentation for each control type)
POSITION	0 to post before pending events, -1 to post after pending events

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.7.4 ATGUICLEAREVENTS

The ATGUICLEAREVENTS routine clears pending events from the event queue. Events for a specific app, form or control can be cleared, and the events to be cleared can by restricted by passing an event mask argument.

Calling syntax:

```
CALL ATGUICLEAREVENTS(APPID, FORMID, CTLID, EVTMSK, MODE, GUIERRORS,
      GUISTATE)
```

Input arguments:

APPID	application identifier (null to clear events from all applications)
FORMID	form identifier (null to clear events from all forms)
CTLID	control identifier (null to clear events from all controls)
EVTMSK	event code mask formed by adding the GE... constants for any events to clear, or zero to clear all events.
MODE	0 = clear events from specified object only 1 = clear events from specified object and any of its descendents 8 = only clear events that have previously been retrieved from the client. Does not query the client for pending events. Only events from the specified object are cleared. 9 = only clear events that have previously been retrieved from the client. Does not query the client for pending events. Events from the specified object and any of its descendents are cleared.

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.8 GUI Macro Functions

1.2.8.1 ATGUIBEGINMACRO

The ATGUIBEGINMACRO routine is called to record a *macro*. Most of the ATGUI… routines may be recorded in a *macro*: all ATGUICREATE… routines, ATGUILOADVALUES, ATGUISETPROP, ATGUISHOW, ATGUIHIDE, ATGUIENABLE, ATGUIDISABLE, ATGUIMOVE, ATGUIRESET, ATGUICLEAR, ATGUIDELETE. When a *macro* is being recorded, instead of performing the desired function, the ATGUI… routines record their parameters in the *macro*. When all desired functions have been recorded, call ATGUIENDMACRO to terminate the recording and return the *macro*. The *macro* may later be “played” by calling ATGUIRUNMACRO. The *macro* may also be stored in a file for future use.

Macros may be used to increase efficiency. It is less efficient to call ATGUISETPROP multiple times than it is to store several calls into one *macro*, then play the *macro*. When many operations are required, such as when creating a form and all its controls, it is much more efficient to use a macro.

While recording a *macro*, it is acceptable to call ATGUIRUNMACRO to add an existing *macro* to the new *macro* being recorded.

Calling syntax:

```
CALL ATGUIBEGINMACRO(ID, GUIERRORS, GUISTATE)
CALL ATGUIENDMACRO(MACRO, GUIERRORS, GUISTATE)
```

Input arguments (ATGUIBEGINMACRO only):

ID	permanent macro ID (necessary for caching on user's PC)

Output arguments (ATGUIENDMACRO only):

MACRO	macro containing all calls since the last ATGUIBEGINMACRO call

Output arguments (both):

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.8.2 ATGUIENDMACRO

The ATGUIBEGINMACRO routine is called to record a *macro*. Most of the ATGUI… routines may be recorded in a *macro*: all ATGUICREATE… routines, ATGUILOADVALUES, ATGUISETPROP, ATGUISHOW, ATGUIHIDE, ATGUIENABLE, ATGUIDISABLE, ATGUIMOVE, ATGUIRESET, ATGUICLEAR, ATGUIDELETE. When a *macro* is being recorded, instead of performing the desired function, the ATGUI… routines record their parameters in the *macro*. When all desired functions have been recorded, call ATGUIENDMACRO to terminate the recording and return the *macro*. The *macro* may later be "played" by calling ATGUIRUNMACRO. The *macro* may also be stored in a file for future use.

Macros may be used to increase efficiency. It is less efficient to call ATGUISETPROP multiple times than it is to store several calls into one *macro*, then play the *macro*. When many operations are required, such as when creating a form and all its controls, it is much more efficient to use a macro.

While recording a *macro*, it is acceptable to call ATGUIRUNMACRO to add an existing *macro* to the new *macro* being recorded.

Calling syntax:

```
CALL ATGUIBEGINMACRO(ID, GUIERRORS, GUISTATE)
CALL ATGUIENDMACRO(MACRO, GUIERRORS, GUISTATE)
```

Input arguments (ATGUIBEGINMACRO only):

ID	permanent macro ID (necessary for caching on user's PC)

Output arguments (ATGUIENDMACRO only):

MACRO	macro containing all calls since the last ATGUIBEGINMACRO call

Output arguments (both):

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.8.3 ATGUIRUNMACRO

The ATGUIRUNMACRO routine plays a *macro* or loads a *template*. *Macros* are created by using ATGUIBEGINMACRO and ATGUIENDMACRO. *Templates* are created by the GUI designer. A *template* (GUI project) may be used to create an application and its associated forms and controls (or any subset) and initialize the properties of the application, forms and controls. Besides creating and initializing applications, forms and controls, a *macro* may also delete applications, forms and controls and may call methods such as ATGUISHOW, ATGUIHIDE, ATGUIRESET, etc.

When a *macro* or *template* is created, it may have a permanent ID assigned. If one has been assigned, then the *macro* or *template* is cached on the user's PC. When ATGUIRUNMACRO is called with a

macro or *template* which has a permanent ID, the cache is checked for a valid copy of the *macro* or *template* and the cached copy is used if it is valid.

Calling syntax:

```
CALL ATGUIRUNMACRO(MACRO, SUBST, GUIERRORS, GUISTATE)
```

Input arguments:

MACRO	macro or template created by the GUI designer
SUBST	macro substitutions (reserved for future use)

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.9 GUI Utility Functions

1.2.9.1 ATGUIPRINT2

The ATGUIPRINT2 routine prints a form or displays the Print Setup dialog.

Calling syntax:

```
CALL ATGUIPRINT2(APPID, FORMID, MODE, GUIERRORS, GUISTATE)
```

Input arguments:

APPID	application identifier
FORMID	form identifier
MODE	mode: 0 = show Printer dialog and print specified form (unless user clicks Cancel). If the user cancels, a warning will be returned in GUIERRORS. 1 = print specified form using current (or default) printer selection 2 = show the Print Setup dialog

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see the Errors topic for details). If the user canceled the print dialog, GUIERRORS<1> = 1, and GUIERRORS<2,5> = GRCANCEL.

Other arguments: for internal use - do not modify

```
GUISTATE
```

Note: ATGUIPRINT2 supersedes ATGUIPRINT.

1.2.9.2 ATGUIHELP

The ATGUIHELP routine displays help for the GUI application. AccuTerm GUI supports three different help schemes: traditional Windows help files, HTML documents, and HTML references.

When using Windows help files (help type 0, .hlp or .chm files), this routine will display a topic from a specified help file. If the help file is null, the application help file is used. Windows help is the simplest help scheme, but requires specialized tools to create the help files, which must be deployed on the client PC (or shared folder on the network). You must assign topic numbers to topics in your help file, and those numbers are specified in the HelpID property of the GUI controls. Topic numbers are required whether you use .hlp or .chm help.

AccuTerm GUI can also display help in the form of HTML formatted text using a dedicated browser window (help type 1). Using this scheme, the host maintains all of the help content, and delivers HTML formatted text to display in response to the application GEHELP event. The HTML text can contain hyperlinks to other host-based help topics using the special TOPIC keyword in the HREF attribute of the <A> (anchor) tag. For example, to create a hyperlink to a topic called "NEWUSER", create an <A> anchor element like:

```
<A HREF="TOPIC:NEWUSER">Help with new Users</A>
```

When the user clicks on the hyperlink, the GUI application will fire a GEHELP event, passing 2 in `GUIARGS<1,1>` and the topic ID ("NEWUWER") in `GUIARGS<1,2>`.

The final help scheme (help type 2) available for AccuTerm GUI applications uses a web server to serve help text as standard web pages. When using this scheme, help topics are actually URLs.

Calling syntax:

```
CALL ATGUIHELP(APPID, TYPE, FILENAME, TOPIC, WINDOW, GUIERRORS,
      GUISTATE)
```

Input arguments:

`APPID`	application identifier associated with the help being displayed
`TYPE`	help type: 0 = Windows help file (.hlp or .chm) 1 = HTML document 2 = HTML reference
`FILENAME`	Windows help file name (help type 0 only)
`TOPIC`	topic number in help file or zero for contents (help type 0), HTML formatted help text (help type 1) or URL of page to display (help type 2)
`WINDOW`	reserved (pass NULL for this argument)

Output arguments:

GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.9.3 ATGUIMSGBOX

The ATGUIMSGBOX routine is called to display a custom message and accept a variety of actions from the user. A simple dialog box is displayed with a custom message and caption and one or more pre-defined buttons. The user must click one of the buttons to dismiss the dialog, and a code is returned to indicate which button the user clicked.

Note: constants shown here are defined in ATGUIEQUATES; the equivalent numeric value can also be used.

Calling syntax:

```
CALL ATGUIMSGBOX(PRMPT, CAPTION, STYLE, BUTTONS, HELPID, RESPONSE,
       GUIERRORS, GUISTATE)
```

Input arguments:

PRMPT	custom prompt message (multiple lines are OK; separate lines with subvalue marks)
CAPTION	dialog box caption text
STYLE	specifies which icon to display in message box: MBNOICON (0) = no icon MBXICON (1) = "X" MBEXICON (2) = "!" MBQICON (3) = "?" MBIICON (4) = "i"
BUTTONS	specifies which buttons to include in message box: MBOK (0) = OK only MBOKCANCEL (1) = OK/Cancel MBABORTRETRYIGNORE (2) = Abort/Retry/Ignore MBYESNOCANCEL (3) = Yes/No/Cancel MBYESNO (4) = Yes/No MBRETRYCANCEL (5) = Retry/Cancel add one of the following to specify which button is the "default" MBDEFAULT1 (100) = first button MBDEFAULT2 (200) = second button MBDEFAULT3 (300) = third button
HELPID	topic in help file to access when user presses the F1 key (this feature is not implemented at this time)

Output arguments:

RESPONSE	indicates which button was clicked: MBANSOK (1) = OK MBANSCANCEL (2) = Cancel MBANSABORT (3) = Abort MBANSRETRY (4) = Retry MBANSIGNORE (5) = Ignore MBANSYES (6) = Yes MBANSNO (7) = No
GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.9.4 ATGUIINPUTBOX

The ATGUIINPUTBOX routine is called to prompt for user input. A simple dialog box is displayed with a custom prompt message and caption. The user can enter a single line response, and click either the OK or Cancel button to dismiss the dialog. If the user clicks the Cancel button, the returned value is null.

Calling syntax:

```
CALL ATGUIINPUTBOX(PRMPT, CAPTION, DEFAULT, HELPID, VALUE, GUIERRORS,
      GUISTATE)
```

Input arguments:

PRMPT	custom prompt message (multiple lines are OK; separate lines with subvalue marks)
CAPTION	dialog box caption text
DEFAULT	initial value to load into input field
HELPID	topic in help file to access when user presses the F1 key (this feature is not implemented at this time)

Output arguments:

VALUE	text string input by user
GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.9.5 ATGUIFILEDIALOG

The ATGUIFILEDIALOG routine is called to prompt for a file name to open or save, or to select a folder. A standard “Open”, “Save As” or "Browse for Folder" dialog box is displayed with a custom caption. If the user clicks the Cancel button, the returned value is null.

Calling syntax:

```
CALL ATGUIFILEDIALOG(CAPTION, DEFAULT, FILTER, STYLE
      VALUE, GUIERRORS, GUISTATE)
```

Input arguments:

CAPTION	dialog box caption text
DEFAULT	initial path or file name to load into input field
FILTER	list of file types and their extensions; syntax consists of type:extn, extn;type:extn …. For example, to allow documents or all files, specify “Document files:*.doc,*.txt;All files:*.*”
STYLE	style: 0 = "Save As" dialog 1 = "Open" dialog - one file 2 = "Open" dialog - multiple files 3 = "Browse for Folder" dialog

Output arguments:

VALUE	file or folder name (multiple file names are separated by value marks)
GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.9.6 ATGUICOLORDIALOG

The ATGUICOLORDIALOG routine is called to allow a user to choose a color. Colors can be selected from standard and system color drop-down lists, or a custom color can be chosen.

Calling syntax:

```
CALL ATGUICOLORDIALOG(COLOR, GUIERRORS, GUISTATE)
```

Input arguments:

COLOR	initial color

Output arguments:

COLOR	chosen color, or NULL if user cancelled
GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see the Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

1.2.9.7 ATGUIFONTDIALOG2

The ATGUIFONTDIALOG2 routine is called to allow the user to choose a font.

Calling syntax:

```
CALL ATGUIFONTDIALOG2(FONTNAME, FONTSIZE, FONTBOLD, FONTITALIC,
      FONTUNDERLINE,
      GUIERRORS, GUISTATE)
```

Input arguments:

FONTNAME	current font name
FONTSIZE	current font size
FONTBOLD	current font bold style (0=normal, 1=bold)
FONTITALIC	current font italic style (0=normal, 1=italic)
FONTUNDERLINE	current font underline style (0=normal, 1=underline)

Output arguments:

FONTNAME	selected font name, or NULL if cancelled
FONTSIZE	selected font size
FONTBOLD	selected font bold style (0=normal, 1=bold)
FONTITALIC	selected font italic style (0=normal, 1=italic)
FONTUNDERLINE	selected font underline style (0 = normal, 1 = underline)
GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see the Errors topic for details)

Other arguments: for internal use - do not modify

```
GUISTATE
```

Note: ATGUIFONTDIALOG2 supersedes ATGUIFONTDIALOG.

1.2.10 List Searching

When searching for an item in a list (list control, combo-box or drop-down list, including in a grid cell), the keyboard can be used to quickly locate the desired item. As each key is typed, it is concatenated to a hidden search string that is used to search the list. The first item in the list that matches the current search string is selected. When a key causes the hidden string to not match any items, the search string is reset to the last key entered and the search is repeated.

For example, if a list contains items 'car', 'cat' and 'dog', typing 'c' would select 'car', typing 'a' would keep the same selection. Typing 't' would change the selection to 'cat'. Typing 'd' would change the selection to 'dog'.

1.2.11 Drag and Drop

AccuTerm GUI supports a simple drag-and-drop mechanism. Basically, every GUI object has `GPDRAGID` and `GPDROPIDS` properties. The `GPDRAGID` property identifies the "drag source", the object being dragged. The DragID property value is an arbitrary string, and could be the same as the object's CTRLID if desired.

Each possible "drop target" (the object under the mouse) has a list of potential DragID values in its `GPDROPIDS` property. As an object is dragged around the form, the source DragID is compared with the DropIDs list of the object under the mouse. If there is a match, the mouse cursor changes to  indicating that the object under the mouse (the "drop target") will accept a "drop" from the "drag source" object. When the IDs do not match, the mouse cursor changes to  . The IDs in the `GPDROPIDS` property are separated by a vertical bar (|) character.

While dragging, if the mouse is released over an acceptable target object, the target receives a `GEDRAGDROP` event. Event arguments indicate the source object ID (the object's ID, not its `GPDRAGID` property), the mouse position in the source object where the mouse button was originally depressed and the mouse position in the target object where the mouse button was released. The mouse coordinates are relative to the upper-left corner of their respective objects. Additionally, a given object type may include other information, such as a grid cell coordinate, list item index, or character position, depending on the specific object type. This applies to both the source and target objects.

It is permissible for an object to be both a "drag source" and "drop target". For example, it is possible for a grid to be its own drag source and drop target, allowing the user to drag and drop cells within the grid.

1.2.12 Error Handling

GUIERRORS is a structure (dynamic array) returned as a result of all ATGUI... routines. The GUIERRORS structure is capable of returning multiple errors, since many of the ATGUI... routines perform multiple tasks (e.g. create control then set several property values).

An error is described by several properties: severity, command, control ID, property code, error number and error description. The first attribute of the GUIERRORS structure, `GUIERRORS<1>`, indicates the maximum severity of any error returned by the call. The following severity levels may be returned:

0 = no error
1 = warning
2 = command failure

3 = fatal error

Fatal errors may not be recovered, and the GUI environment should be shut down and the application terminated. The handling of command failure and warning errors is up to the discretion of the programmer. In some circumstances it may be appropriate to ignore these errors; at other times it may be appropriate to terminate the application.

All errors are returned in the following attributes of the GUIERRORS structure. Each error (attribute) consists of 6 fields (separated by value marks).

`GUIERRORS<x,1>` = error severity

`GUIERRORS<x,2>` = command code (see `GC…` constants in ATGUIEQUATES)

`GUIERRORS<x,3>` = ID of app/form/control which encountered the error

`GUIERRORS<x,4>` = property code (see `GP…` constants in ATGUIEQUATES) if error involved a property

`GUIERRORS<x,5>` = error number

`GUIERRORS<x,6>` = error description

Note: most of the GUI library routines call ATGUIERRORS to process errors returned from the runtime engine. The result of this processing is returned in the GUIERRORS argument. ATGUIERRORS and ATGUIERROR are not intended to be called by application programs.

1.3 The GUI Designer

The GUI designer is used to create and maintain GUI projects. The GUI project contains all information required to create a GUI application including all of the forms and all of the constituent controls (fields) and the initial value of most properties. The project record is stored in the format required by the GUI Library ATGUIRUNMACRO subroutine. Header information is included in the project record to support caching of the project on the workstation.

The GUI designer allows the visual design of forms, similar to the Microsoft Visual Basic development environment. All supported controls and most of the controls' properties can be created and modified in the designer.

The designer is invoked by the **GED** verb from TCL. The **GED** verb syntax is similar to the standard **ED** verb:

GED *filename itemid*.

The **GED** design environment consists of a main window divided into two panes. The currently selected form is displayed in the left pane, and a project tree is displayed in the right pane. A splitter bar can be used to resize the two panes. On the far left, a control palette displays an icon for each GUI control which can be created. The top icon creates a new form. Clicking on any icon opens a properties dialog where you assign control (or form) properties such as the ID, caption, colors, fonts, etc. After specifying any desired properties, click the **Create** button to create the control (or form).

To change the location of a control on a form, click on it in the form pane and drag it to the desired position. To resize a control, click and drag one of the "nibs" at the edge of the selected control.

When the project is executed by the GUI runtime, each element is created in the same order as the elements in the project tree. This is also the "tab order" (see Tab Order topic for more information). You can adjust the order of the items in the project tree by dragging and dropping within the project tree. Items may only be dragged to change their order; they cannot be moved to other forms or container controls by dragging – use cut & paste for this.

The GUI designer will allow you to have more than one GUI project open at once. This is often useful when you would like to cut, copy and paste elements that you have designed from one project into another.

When you are finished creating or modifying a GUI project, use **File ► Save** from the main menu or click the  button on the toolbar to save the updated GUI project back.

GUI Program Code

Each GUI project has an associated program code item that handles the startup and shutdown of your GUI application. While the application is running, the program code processes *events* that are triggered by users interacting with the application. Things like clicking a button or tabbing from one field to another are transformed into an *event* and sent by the AccuTerm GUI Runtime to the program for processing. The program runs an *event loop* which retrieves *events* from AccuTerm GUI. The program then decodes the *event* using a code section called the *event decoder.* The *event decoder* then calls an *event handler* subroutine that you (the programmer) supply to process the event. Generally, you (the programmer) only need to write the *event handler* subroutines to provide the functions particular to your application.

The GUI Designer is integrated with the wED code editor. You use the **wED** editor to edit the MultiValue BASIC program code associated with the GUI project you are working on. Use **Tools ► Edit Code** from the main menu or click the  button on the toolbar to open the program code. The program code is opened as a new document in the wED code editor. If the program code is already open, the wED editor window is activated.

Program Code Template

The first time you access the code tools without an associated program file and item ID (the program code), the GUI designer can generate a new skeleton program for you. The **Code Generator** uses code templates to generate the program code. The AccuTerm GUI Designer includes code templates for "standalone", "subroutine" and "dialog" applications. You can create your own code templates to customize the code generation process (use **View ► Preferences** from the main menu to change the folder where the code generator looks for templates, or add your templates to the GuiLib folder.)

Standalone: this code template will create a self-contained BASIC program for the GUI application. When running programs created using this template, you cannot run more than one GUI application at a time. Using the EXECUTE statement to start a second GUI program from an event handler in another program is not possible (at least not without some tricky modifications to the code).

Dialog: the dialog code template creates a callable subroutine that displays dialog-style forms modally. Only dialog-style forms may be used in this code model. While a modal dialog form is open, the user cannot switch to another form in the application until the current form is dismissed. A dialog form can open another dialog form. Subroutines created with this template are callable from any GUI or green-screen application, and it is possible to call another dialog subroutine from an event handler in a different subroutine.

Subroutine: the subroutine code template creates a callable subroutine which is designed to cooperate with other subroutines generated from the same template. Multiple GUI application can be run concurrently using the subroutine template. The subroutine must be called by a main dispatch program patterned after the ATGUI.MAIN.SAMPLE sample program. You need to provide a mechanism to preserve the state of the subroutine when control is passed to a different GUI application. See Multiple Application Model for more details on using GUI subroutines.

MDI Subroutine: this is similar to the Subroutine template described above, except that it is used when developing an MDI Multiple Application Model application. There must be a master MDI parent container project and one or more MDI child projects, each using this template.

Event Handlers

To make it easy to add or modify *event handler* subroutines in the program code, the design form and project tree provide a context (right-click) menu with an **Edit Code** menu selection. Clicking **Edit Code** from the context menu will show a sub-menu with a selection for each event you have selected for the object in the Events tab of the Properties dialog. Selecting an event from this list will insert the event into your program (if it does not already exist), then scroll the program code so that the beginning of the event subroutine is in view and activate the wED editor.

Use the **Update Code** tool (from the **Tools** menu or  toolbar button) to check if the program code associated with your GUI project is up to date. The **Update Code** tool can update the event decoder, insert missing event handlers and remove unused event handler code from the program. *If you add or*

remove controls, or change the selected events for a control, the event decoder section of your GUI program will need to be rebuilt. This condition is indicated in the status bar at the bottom of the designer window. Use the **Update Code** *tool to rebuild the decoder when this happens.*

1.3.1 Opening a GUI Project

To open a GUI project in the designer, select use **File ► Open** from the main menu. If you are connected to your MultiValue host (using the GED verb), enter (or browse to) the desired host file name and item-ID for the GUI project you want to open. If you want to open a project saved on your PC's file system, click the *My Computer* icon and browse to the desired directory, then double-click the desired file name.

1.3.2 GUI Object Properties

The **Properties** dialog is used to specify the properties or characteristics of the selected control or form. Properties which can be specified include the control ID, caption, colors, font and events to respond to.

You can open the properties dialog for any control (or form) in several ways:

- Double-click on the control in the form pane
- Select the control (by clicking on it) and press the F4 key
- Select the control then use **Edit ► Properties** from the main menu
- Double-click on the control ID in the project tree
- Right-click the control in the form and select **Properties** from the popup menu
- Right-click the control ID in the project tree and select **Properties** from the popup menu.

The **Properties** dialog is arranged using index tabs to organize the properties into logical categories. Not all categories are available for every control type.

Category	Applies to
General	all
Application	application
List	list and combo-box controls
Grid	grid controls
Options	option-group control
Events	all
Colors	forms and most controls
Font	forms and most controls
Picture	forms, command button & picture box
Icon	application and forms
Tabs	tab controls
Tree	tree controls
Drag & Drop	all controls
Menu	menus and toolbars

1.3.2.1 General Properties

The *General* tab of the Properties dialog is available for all GUI objects: applications, forms and controls. The specific properties which are available in this tab vary according the type of object currently selected in the main design form.

ID
For all GUI objects, the *ID* is displayed and can be modified. If the object is an Application, then the **App Name** may be modified; if it is a Form, then **Form ID** can be adjusted; if it is a control, then the **Control ID** can be changed.

Style
Often, a given object type will have a number of available *Styles*. For example, an Application may use the Single Document Interface or Multiple Document Interface; a form can be fixed, sizable or a dialog box; a list control can be single or multi selection or be a drop-down list; a command button can be normal, OK or Cancel; a grid can be protected or editable. etc.

Scale
The Application object is the only object with a *Scale* property. This property specifies the measurement used for all size and location coordinates for all forms and controls that belong to the application. The default scale mode is "characters". The exact conversion from characters to pixels is dependent on monitor resolution and system font size. AccuTerm GUI considers a character to be 12 points high by 6 points wide.

Position & Size
Although the position and size of the object may be adjusted by modifying the *Left*, *Top*, *Width* and *Height* values, it is often easier to reposition and resize controls by dragging on the designer form. The position of a Form (or MDI Application) object may be set to "auto" to cause the form to display centered on the screen.

Window State
For sizable Form objects, specifies the initial window state as "Normal", "Minimized", or "Maximized".

Caption
Forms, labels, check-box, option group, command button and frame controls have a *Caption* property. The caption may contain an "&" (ampersand) character to create a keyboard shortcut. For example, if the caption is &Help, the text will display as Help, and pressing the **Alt+H** keys will activate the control. In the case of a Label control, the keyboard shortcut will activate the next control in the project tree (usually an edit or combo control).

Visible
This setting determines whether a form or control is visible. The *Visible* property may be changed by code at runtime to show or hide any form or control. *Contrary to normal design principles, changing the Visible property of a Form may trigger Activate and Deactivate events for the form that becomes active or inactive. There may also be Activate, Deactivate and Validate events triggered for the control being activated or deactivated when the active form changes.*

Enabled
This setting determines whether a form or control is enabled. When a control is disabled, it is normally displayed in a light-gray color to indicate to the user that it is disabled. Disabled controls cannot be activated or otherwise updated by the user.

Tab stop

Normally, controls have the *Tab stop* property set. However there are times that the application developer may want certain controls to be "skipped" when tabbing through the fields on a form. Un-checking this property will cause the control to be skipped. See also: tab order.

Read only
Setting the *Read only* property of a control prevents the user from modifying the value of the control. This property applies to edit, combo, list, check and option group controls.

Required
Check the *Required* check box if a value in an edit, combo, list or option group control is required.

Word wrap
Multi-line edit controls normally have the Word Wrap setting enabled. Uncheck this option to disable word wrapping in the edit control. This will cause the control do display both vertical and horizontal scroll bars.

Grid controls normally do not have Word Wrap enabled. Check this option to support word wrapping in grid cells.

Drag mode
If the control is a grid, select the desired drag mode. This defines the grid behavior when dragging the mouse around the grid. Choices are to change the active cell, select cells, initiate drag-and-drop, or ignore.

Focus style
If the control is a grid, select the desired focus indicator: thin dotted line, thick dotted line, solid line or none.

Help topic ID
The *Help topic ID* field is used to associate a Windows Help file topic number with a form or control. When using the Windows help interface to supply online help for your GUI application, the topic ID for the active form or control is used to display context sensitive help regarding the active form or control.

Help hint text
The *Help hint text* field is used to provide a help text message when the user hovers the mouse over the control. *Note: while the Hint property supports multi-line help hint text, you can only enter a single line hint using the GUI designer. Multiline hints must be assigned at run time using* ATGUISETPROP*.*

Border style
Many controls have a *Border style* property. This property can be set to none (no border), flat (thin black outline) or normal (Windows' native border style). Controls which have a *Border style* property are label, edit, list, combo, option group, grid, tree,frame and picture.

Alignment
Label and edit controls have an *Alignment* property. Text in these controls can be left or right justified or centered.

Default value
Edit and combo controls have a *Default value* property. This is the value which is loaded into the control when the control is initially created or when the Form is reset.

Default state
Check-box controls have a *Default state* property. The Default state is the state (checked or unchecked)

of the Check-box when the control is initially created or the Form is reset.

Data type
Edit and combo controls have a *Data type* property. Setting this property to "Any" allows any value to be accepted as valid; specifying any other data type causes the value to be validated locally, and if the value does not match the selected data type, the user will be prompted to re-enter the field.

Maximum Length
Edit and combo controls have a *MaxLen* property. If you need to limit the number of characters to accept, specify that value here.

Maximum Lines
Multi-line edit controls have a *MaxLines* property. If you need to limit the number of lines the user can enter, specify that value here.

1.3.2.2 Application Properties

The *Application Properties* tab is available only when the currently selected object is an Application. This tab allows you to enter certain properties which apply to the entire GUI application.

Application title
The application title is shown in the MDI title bar if the application style is MDI.

Cache ID
Setting a *Cache ID* allows the GUI runtime to cache the GUI project on the user's hard disk. This improves performance when starting a GUI application. The cached template includes a time/date stamp and is automatically updated when you use the GUI designer to update the template.

Help type
AccuTerm GUI supports three help schemes: Windows help files (.hlp or .chm), HTML Documents (using the Help event) and HTML References (specified by a URL). If you select Windows help files, you will need to create a Windows help file (.hlp or .chm) and deploy it to the client PC. If you use HTML Document help, the application will trigger a Help event when the user requests help. If you use HTML References help, help text will be retrieved from a web server using the specified URL.

Help file name
If you select Window help files for the Help Type, enter the name of your help file in this field.

Help index URL
If you select HTML References for the Help Type, enter the URL of the "index" page in this field.

Description
The *Description* field is displayed in the "About box" for your application. Use "*ABOUT" as the menu *Tool ID* to display the "About box".

Copyright
The *Copyright* field is displayed in the "About box" for your application. Use "*ABOUT" as the menu *Tool ID* to display the "About box".

Logo file name
Specify a bitmap or JPEG file in the *Logo file name* field to display a custom logo in the "About box". The image is resized to approximately 292 x 58 pixels.

Author, Version
The *Author* and *Version* properties are text fields which are displayed in the "About box" for you application. Use "*ABOUT" as the menu *Tool ID* to display the "About box".

1.3.2.3 Application Traits

The *Traits* tab is available only when the currently selected object is an Application. This tab allows you to customize the behavior of several facets of a GUI application.

Host busy message & timeout
When a GUI application is waiting for the host to process an event, the user interface is non-responsive. This is by design, as the host program can modify the state of any GUI objects while processing an event, changing the outcome of future events. If the host is going to be busy processing an event for a noticeable length of time, the GUI runtime engine will display a "busy message" panel to inform the user. The default message is "Processing... please wait".

The default time delay after an event is retrieved by the host before the "busy message" is displayed is 2 seconds. You can adjust this to any desired value.

Setting the timeout to zero will prevent the "busy message" panel from being displayed at all.

Behavior of the Enter key
There are three settings for this option:

Normal - the Enter key adds lines to *multiline edit controls*, actuates (clicks) an active *command button* or actuates the "OK" button, if one is present.

Same as Tab key - the Enter key works just like the Tab key and activates the next field in the tab order. The "OK" button is not supported.

Same as Tab, except - the Enter key activates the next field, unless a *multiline edit control* or *command button* is active. The "OK" button is not supported.

Auto select text fields
If this setting is checked, when an *edit control* is activated by the Tab or Enter key, the current value is selected. When the value is selected, typing a new value replaces the current value. When this setting is unchecked, the current value is not selected upon activation, and you need to manually delete the current value before entering a new value.

Disable tree control automatic tooltips
Normally, if the text for a node in a tree control is wider than the control, a tooltip with the full node text is displayed over the node so that the user can read the full text of the node. If this setting is checked, these automatic tooltips are disabled.

1.3.2.4 List Properties

The *List Properties* tab is available for *List-box* and *Combo-box* controls. This tab provides the ability to define list columns, column widths, column alignment as well as the initial list contents. You can add or delete columns, select grid lines and set column headings.

The List properties are organized into two categories: Columns and Items.

1.3.2.4.1 List Properties - Columns

The Columns tab of the List Properties allows you to add or delete columns, define column widths and alignment, designate column headings and choose the grid line style. A sample list showing current column headings is displayed at the bottom of the page.

Column
Use the *Column* up/down buttons to select the current column and number of columns. The current column can also be selected by clicking in the desired column in the sample list.

Data type
Each column has a *Data type* property. This property is used only for sorting the list (ATGUISORT). If a column containing date or time data will be sorted, the *Data type* property must set appropriately.

Alignment
The *Alignment* drop-down displays the column alignment of the currently selected column. To change column alignment, drop down the list and select a new setting.

Width
The *Width* field displays the width of the currently selected column. The measurement units are defined by the *Application* object. You can enter a new width into this field, or you can use the mouse to drag the column boundry left or right to adjust the column width.

Data column
For a combo or drop down list control, you can select the column that is copied to the edit portion of the control when a list item is selected. This is "data column".

Show icon in first column
List controls can display an icon in the first column of the list. Each item can have a different icon. If you enable this option, you will need to set the icon filename or URL `GPICON` property using ATGUISETPROP.

Icon size
If icons are being used, select the icon size (small or large) from this drop-down list.

Add column and **Insert column**
The *Add* and *Insert column* buttons create a new column. The newly created column becomes the "selected" column.

Delete column
Click the *Delete column* button to delete the current column.

Show column headings
Check the *Show column headings* check-box to display column headings. After setting this option, you

can click in the column heading area of the sample list and enter your heading text.

Grid lines
Choose the grid line style from the *Grid lines* drop-down. Available styles are none, horizontal, vertical or both.

1.3.2.4.2 List Properties - Items

The *Items* tab of the *List Properties* lets you enter the initial item list.

Row
Use the up/down buttons to select the current *row* and number of rows. Alternatively, you can click on any row in the sample list to select that row. Row 0 is the "heading" row; row 1 is the first list item.

Max dropdown lines
For combo and drop-down list controls, you can specify the number of lines to display in the drop-down list. The default is 8 lines.

Default row
For single-selection list controls, one of the items can be designated the "default row". This item will be initially selected when the project is loaded. If no items have the default row option checked, no items will be initially selected.

Add row and **Insert row**
Click the *Add* or *Insert row* button to create a new row.

Delete row
Click the *Delete row* button to delete the current row.

Items
Enter list *items* in the sample list. Click in the desired cell (row & column) and enter the item text.

If the "show icon in first column" option is enabled, when a cell in the first column is selected, an "icon selector" tool will be displayed just below the current cell. Click the browse button to browse for an icon file, or enter a filename or URL for the item's icon.

1.3.2.5 Grid Properties

The *Grid Properties* tab is available for *Grid* controls. This tab provides the ability to define grid columns, column widths, column alignment, column type, data type, drop-down lists for drop-down columns, as well as the initial grid contents. You can add or delete columns, select grid lines and set column headings.

The Grid properties are organized into three categories: Columns , Items and Dropdown List.

1.3.2.5.1 Grid Properties - Columns

The *Columns* tab of the Grid Properties allows you to add or delete columns, define column widths and alignment, designate column headings, column type, data type and choose the grid line style. A sample

grid showing current column headings is displayed.

Column
Use the *Column* up/down buttons to select the current column and number of columns. The current column can also be selected by clicking in the desired column in the sample list.

Locked columns
If desired, the leftmost columns of a grid can be "locked" so that they do not scroll. Enter the desired number of locked columns in this field.

Column type
For each column you can designate a *Column type*. When a grid cell is activated at runtime, the cell is displayed according to the selected *column type.* Choose from *Label, Text box, Check box, Dropdown list, Dropdown combo, Label+ellipses, Label+icon* or *Label+button.* If you select *Label* (any variation), then the user will not be able to enter data into cells in this column. *Text box* allows free-form text entry. *Check box* allows the cell to be "checked" or "unchecked". *Dropdown list* columns allow the user to select one of several pre-defined choices. *Dropdown combo* columns allow the user to select one of several pre-defined choices or enter free-form text. *Label+ellipses* and *Label+button* display a small button at the right edge of the cell when the cell is active. Clicking the button fires a BtnClick event. *Label+Icon* is the same as a *Label* column except that an icon is displayed at the left edge of the cell.

Read only
If the *Read only* option is selected, then the column will not allow the user to change the contents of any cell in the column. A *text-box* column with *read only* selected is identical to a *label* column.

Alignment
The *Alignment* drop-down displays the column alignment of the currently selected column. To change column alignment, drop down the list and select a new setting.

Width
The *Width* field displays the width of the currently selected column. The measurement units are defined by the *Application* object. You can enter a new width into this field, or you can use the mouse to drag the column boundary left or right to adjust the column width.

Sizable
If the *Sizable* option is selected, the user will be allowed to resize the column at runtime by dragging the column separator with the mouse. A Resize event will be fired if the user resizes a column.

Data type
Each column has a *Data type* property. Setting this property to "Any" allows any value to be accepted as valid; specifying any other data type causes the value to be validated locally, and if the value does not match the selected data type, the user will be prompted to re-enter the cell. The *Data type* property is also used when the grid is sorted (ATGUISORT). For proper sorting of date and time data, an appropriate data type is required.

Max length
Enter the maximum number of characters to accept for text or combo columns, or zero for no limit.

Required
Check the *required* check-box if data in a text or combo column is required.

Column help hint text
Each column can display help hint text when the mouse hovers over the column heading.

Add column and **Insert column**
The *Add* and *Insert column* buttons create a new column. The newly created column becomes the "selected" column.

Delete column
Click the *Delete column* button to delete the current column.

Show column headings
Check the *Show column headings* check-box to display column headings. After setting this option, you can click in the column heading area of the sample grid and enter your heading text.

Grid lines
Choose the grid line style from the *Grid lines* drop-down. Available styles are none, horizontal, vertical or both.

1.3.2.5.2 Grid Properties - Items

The *Items* tab of the *Grid Properties* lets you enter the initial item list.

Row
Use the up/down buttons to select the current *row* and number of rows. Alternatively, you can click on any row in the sample list to select that row. Row 0 is the "heading" row; row 1 is the first list item.

Automatically add new lines to grid
If you select this option, when the user enters data on the last line of the grid, a new blank line will automatically be added to the grid.

Add row and **Insert row**
Click the *Add* and *Insert row* button to create a new row in the grid.

Delete row
Click the *Delete row* button to delete the current row.

Items
Enter grid *items* in the sample grid. Click in the desired cell (row & column) and enter the item text, check-box state, or select an item from the drop-down list.

When a cell in a column type of *Label+icon* or *Label+button* is selected, an "icon selector" tool will be displayed just below the current cell. Click the browse button to browse for an icon file, or enter a filename or URL for the cell's icon.

1.3.2.5.3 Grid Properties - Dropdown List

The *Dropdown List* tab allows you to enter the pre-defined choice list for the currently selected column (the column selected in the *Columns* tab). This tab is only available if the selected column's *Column type* is *Dropdown list* or *Dropdown combo*.

Columns
Use the up/down buttons to select the current column to edit, or to add or remove columns from the

drop-down list.

Rows
Use the up/down buttons to change the number of rows in the pre-defined drop-down list for the selected column.

Enter the items to populate selected column's drop-down list. Click on a cell to activate it. Use the down-arrow key to add a new row to the grid. The grid columns will auto-size as you enter item data.

If the drop-down list itself has multiple columns, you can select which column is the "data column" by checking the **Data column** check-box for that column.

1.3.2.6 Option Properties

The *Option Properties* tab is used to define the items in an *option group* control.

Button arrangement
The option buttons can be arranged with the buttons going vertically or horizontally. When the buttons are arranged vertically (down), the first button is at the top, the second button under the first, etc. When arranged horizontally (across), the first button is to left, and the second button to its right. If there are more buttons than fit a single column or row, the buttons are automatically arranged into additional columns and row.

Columns
Use the *columns* up/down buttons to choose a fixed number of columns to arrange the options into or select *auto* to automatically determine column layout.

Rows
Use the *rows* up/down buttons to choose a fixed number of rows to arrange the options into or select *auto* to automatically determine row layout.

Add option and **Insert option**
Click the *Add* and *Insert option* button to create a new option button.

Delete row
Click the *Delete option* button to delete the current option button.

Options
The options grid displays each of the option buttons for the option group control and allows you to enter the option caption and select whether the option is **enabled**. You can also select **default** for a single option by checking the appropriate box. If desired, enter **help hint** (tooltip) text for individual options.

1.3.2.7 Event Properties

The *Event Properties* tab is used to select which *events* you want your application to process. Unless an event is selected in this tab, your application will not be notified when the event occurs. Consult the following table to find out if an event is supported by a particular control type.

Event	Description	Supported Controls
Close	The user has clicked the form's close button (X). To close the form, call ATGUIDELETE passing the form and application ID.	Application (MDI), Form
Activate	The Activate event is triggered when the user activates a new form (by clicking on the form or from the taskbar). It is also triggered when the user activates a control by clicking on it or by pressing the Tab key. This event follows the Deactivate event.	Application, Form, Edit, List, Combo, Option group, Check box, Grid, Command button, Tab group, Tree
Deactivate	The Deactivate event is triggered when the user activates a new form (by clicking on the form or from the taskbar). It is also triggered when the user activates a control by clicking on it or by pressing the Tab key. This event precedes the Activate event.	Application, Form, Edit, List, Combo, Option group, Check box, Grid, Command button, Tab group, Tree
Click	The Click event is triggered when the user clicks the mouse left button in a control. It is also triggered when the user selects an item in a List, Combo, option group, tab group, or tree control, and when the state of a check box is toggled, even if the action is preformed using the keyboard instead of the mouse.	Edit, Label, List, Combo, Option group, Check box, Command button, Grid, Tree, Picture, Frame, Tab group, Tab, Menu, Toolbar, Status bar
DblClick	This event is triggered when the user double-clicks on the control.	Edit, Label, List, Combo, Grid, Tree, Picture, Frame, Status bar panel
ColClick	The user has clicked on a column heading	List, Combo, Grid
Context	The Context event is triggered when the user clicks a control with the right mouse button or presses the `MENU` key on the keyboard.	Application (MDI), Form, Edit, Label, List , Combo, Option group, Check box, Command button, Grid, Tree, Frame, Picture, Tab group, Tab, Status bar panel
Change	The Change event is triggered when the contents of an Edit control (or the edit part of a Combo control) is changed. For Edit controls, each change (keystroke) causes this event to fire. For Grid controls, the Change event fires after editing a cell or toggling the cell check-box, or when the selection in a drop-down list changes.	Edit, Combo, Grid (editable only)
Validate	The Validate event is triggered when the user has changed the value of a control and is activating a different control (by clicking or pressing the Tab key).	Edit, List, Combo, Option group, Check box, Grid, Tree
ValidateCell	The user has finished editing the value of a cell in a grid. This event is only fired if the cell value has changed.	Grid (editable only)
ValidateRow	The user has changed the current grid row by clicking on a different row or by using the cursor	Grid (editable only)

Event	Description	Supported Controls
	keys.	
BtnClick (formerly Ellipsis)	The user clicked the cell button (...) in a grid.	Grid
ActivateCell	The user has changed the current grid cell (column and/or row).	Grid
DeactivateCell	The user has changed the current grid cell (column and/or row).	Grid
ActivateRow	The user has changed the current grid row.	Grid
DeactivateRow	The user has changed the current grid row.	Grid
Status	For Tree controls, the user expanded or collapsed a tree node. For Command buttons, the user pressed or released a mouse button.	Tree, Command button
DragDrop	The user has dropped a control (drag-source) onto another (or the same) control (drop target).	All
Timer	The timeout interval has expired.	Timer
Resize	The user has resized, minimized or maximized the form. Grid: the user has resized a grid column.	Application (MDI), Form, Grid
Help	The user pressed the F1 key, clicked on the *Help menu item or clicked on a TOPIC hyperlink in the help window.	Application
Quit	The GUI application has terminated.	Application

1.3.2.8 Color Properties

The *Colors Properties* tab is used to select foreground (text) and background colors for a *Form, Control,* or *MDI Application*. For *Grid* controls, you can also select an "alternate row" color, which is used as the background color for even-numbered rows.
Some controls (Label, Option group, Check box and Frame) support transparent background.

Color
Select the foreground, background or alternate color from the color drop-down list. The list contains 16 standard colors, followed by Windows system colors. Finally, clicking on the last item, "Custom color >>" will open a color selection dialog to choose a custom color.

Use default color
Check the *Use default color* check-box if you want to select the default color for either the foreground or background. For
controls, the default color is usually the foreground or background color of the parent form or container. For forms, the default colors are determined by the current desktop display properties: the default foreground color is "Window Text", and the default background color is "3D Face".

Background is transparent
Check the *Background is transparent* check-box if you want the control's background to be transparent. This is useful if the container is a Form or Picture control which is displaying an image.

1.3.2.9 Font Properties

The *Font Properties* tab is used to select a font for *Controls* and *Forms.* Although a form does not actually display text, the form's font is used as the default font for all controls on the form. Changing any of the form's font properties will also change the font properties of all controls using the default font.

Use default font
Check the *Use default font* check-box to use the default font. For *controls*, the default font is the form's font. For *forms*, the default font is Arial 8pt Regular.

Change
Click the *Change* button to choose a different font.

Style
If the default font is not selected, you can use the *Style* check-boxes to select various font styles.

1.3.2.10 Picture Properties

The *Picture Properties* tab allows you to select an image file (or URL) to display in a *Picture* control, or as a *Form* or *Command button* background. When you select an image file or URL, a sample of the image is displayed.

Picture formats supported include Picture formats supported include BMP, JPEG, GIF, PCD, PCX, PICT, PNG, PSD, TARGA, TIFF, WBMP, XBM, XPM and Windows Metafile.

If you already have an image selected but no longer want to use it, you can click the *Clear Picture* button.

Pictures are rendered using the FreeImage Open Source image library.

1.3.2.11 Icon Properties

The *Icon Properties* tab allows you to select an icon or image file (or URL). Icons are used in the *Application* and *Form* "control menu" (the icon in the upper-left corner of the window). Icons can also be displayed on *Command buttons* and on *Tabs.* Icons used in list items or grid cells are selected in the List Items or Grid Items tab, not in the Icon tab. If you select an icon for the application, the selected icon becomes the default icon for any forms in that application, and is used unless a form has its own icon set. It is displayed in the **ALT+TAB** window when switching Windows applications.

Picture formats supported include ICO, BMP, JPEG, GIF, PCD, PCX, PICT, PNG, PSD, TARGA, TIFF, WBMP, XBM, XPM (ICO is the preferred format, as it supports transparency and partial transparency.)

For Forms, select the *Show form icon* option if you want to show an icon in the Form's control menu.

Select the *Default icon* option to use the default AccuTerm GUI Form icon  .

If you already have an icon selected but no longer want to use it, you can click the *Clear* button.

1.3.2.12 Tab Properties

The *Tab Properties* tab allows you to set up *Tab group* and *Tab* controls. The *tab group* control is used to organize a form into several categories identified by an "index tab". Normally, the *tab group* control only contains *tab* controls, which then contain other controls. To add a control to a specific *tab* control, first click on the desired tab, then click the control you want to add from the control palette.

It is possible to create controls directly on the *tab group* control by selecting the *tab group* control in the project tree, then clicking on a control from the control palette. These controls are then visible on all *tab* controls within the *tab group.* This is useful for controls such as OK and Cancel command buttons that affect all tabs.

Tab
Use the *tab* up/down buttons to select the current tab or add new tabs. You can also click on a tab in the sample *tab group* to select a tab.

Add tab or **Insert tab**
Click the *Add* or *Insert Tab* buttons to create a new *tab*. *Add* creates the new tab after the last tab; *insert* creates the new tab before the currently selected tab.

Delete tab
Click the *Delete selected tab* button to delete the currently selected tab.

Tab ID
Enter the ID of the selected *tab* control in the *Tab ID* field. The *Tab ID* is used if you need to access the properties of the *tab* control at runtime.

Tab Caption
Enter the text of the *caption* for the selected *tab.* Use an ampersand (&) in the caption to create a shortcut key to the selected *tab*. For example, to use **ALT+L** as a shortcut for your List tab, enter the caption as "&List".

Tab icon
If you want to display an icon on the selected tab, enter the icon filename (or URL), or click the Browse button.

Tab help hint
To display a help hint (tooltip) for the selected tab when the mouse hovers over it, enter the hint text.

Enabled
This setting controls whether the selected tab is enabled or disabled. A disabled tab cannot be selected at runtime.

Visible
This setting controls whether the selected tab is visible or not.

When tabs do not fit across, display multiple tab rows
This setting controls whether tabs are displayed with scroll arrows at the upper-right corner, or whether multiple rows are tabs are displayed instead.

1.3.2.13 Tree Properties

The *Tree Properties* tab is used to define the items in a *tree* control.

The tree property page is divided into two panes: the left pane displays the current tree hierarchy, and the right pane displays the properties of the currently selected tree node.

Node ID
Each node in the tree has a Node ID. This ID must be unique within its particular branch in the tree. The ID is any text string. The backslash character is reserved as a path separator and cannot be used in the Node ID.

Caption
Enter the caption text for the currently selected node.

Node initially selected
Check this box if the currently selected node is the "default" node.

Node expanded
Check this box if the node should be expanded when initially displayed.

Icon
Select an icon for the node.

Node help hint text
Enter the help hint text for the currently selected node.

Click the **Delete node** button to delete the currently selected node from the tree. Any child nodes of the currently selected node will also be deleted.

Click the **New sibling node** button to create a new node at the same level as the currently selected node.

Click the **New child node** button to create a new node that is a child of the currently selected node.

1.3.2.14 Menu Properties

The *Menu Properties* tab is used to define the items in a *main menu*, *popup menu, toolbar* or *status bar*.

The menu property page is divided into two panes: the left pane displays the current menu or toolbar hierarchy, and the right pane displays the properties of the currently selected node.

Toolbar Properties

Style
Select whether the toolbar uses large or small icons.

Position
Select the *docking position* to specify which edge of the window the toolbar is attached to, of if the

toolbar should be "floating".

Visible
Check the *visible* check-box if the toolbar should be visible.

Menu and Toolbar Item Properties

Item ID
When the user clicks a menu item or toolbar button, the *Item ID* is passed to the host program as an argument to the menu Click event (if you use the code generator to build the host program, the generator will create an event handler for each menu or toolbar item, so you do not need to process this argument). The ID must be unique within the scope of the menu or toolbar, and is not case sensitive.

Certain pre-defined IDs can be used to perform common functions (without invoking any host code). The special IDs are: *CUT, *COPY, *PASTE, *SELECTALL, *PRINT, *PRINTSETUP, *CLOSE (MDI child form), *EXIT, *TILE (MDI child windows), *CASCADE (MDI child windows), *WINDOW (this will display a sub-menu which contains items for each visible MDI child window, *HELP and *ABOUT. All of the special IDs begin with an asterisk (*), and can be selected from the drop-down list.

Caption
The *Caption* is displayed in the menu. For top-level menu items, you can add a keyboard "shortcut" by inserting an ampersand (&) in the caption text immediately before the desired letter. For example, if the menu caption text is “Close”, and the desired shortcut key is “C”, then the caption property should be set to “&Close”. To use the keyboard shortcut, the user presses the shortcut key while the Alt key is depressed (e.g. **ALT+C**).

Shortcut
It is possible to associate a keyboard shortcut with a menu item. This shortcut is normally used with control-keys, like defining Ctrl+C as a shortcut for Copy. If you want to associate a shortcut with a menu item, select the desired key from the dropdown list. Note: this is how to link an action (event) to a function key in the GUI environment.

Help hint text
Enter any desired text to display as a help hint (tooltip) when the mouse hovers over the toolbar button or status bar panel.

Visible
Check the *visible* check-box if the menu item, toolbar button or status bar panel should be visible. This is normally the case.

Enabled
Check the *enabled* check-box if the menu item, toolbar button or status bar panel should be enabled. This is normally the case.

Begin group
Check the *begin group* check-box if you want a separator to precede the menu item or toolbar button.

Causes validation
Check the *causes validation* check-box if you want to trigger a *validate* event then the item is clicked.

Icon
Each toolbar button must have an associated icon. Menu items and status bar panels may also have an

associated icon, but it is not required. To choose an icon for a menu item, toolbar button or status bar panel, enter the file name or URL of the icon in the icon *file name* box, or click the *browse* button to select an icon file.

Click the **Delete item** button to delete the currently selected item from the tree. Any child items of the currently selected item will also be deleted.

Click the **New item** button to create a new menu item or toolbar button node at the same level as the currently selected item.

Click the **New sub-menu** button to create a new menu item that is a sub-menu of the currently selected item.

1.3.2.15 Drag and Drop Properties

The *Drag and Drop Properties* tab allows you to drag and drop behavior of a control.

AccuTerm GUI supports a simple drag-and-drop mechanism. When a control is being dragged (the "drag source"), AccuTerm GUI compares it's DragID to a list of DropIDs that belong to the control the mouse is currently over. If the IDs match, a "drop" is permitted, as indicated by the mouse cursor. When the mouse button is released over an acceptable control (the "drop target"), the "drop target" receives a DragDrop event.

To enable a control to be a "drag source", enter a Drag ID for the control. This is an arbitrary string, but could be the Control's CTRLID (check the *Use Control ID as Drag ID* check-box to automatically set the Drag ID.

To enable a control to be a "drop target", check the *Drop target* check-box and add one or more IDs to the Drop IDs list. You can enter an arbitrary ID and click the *Add* button, or select one or more from the Drag ID list and click the *Add* button. To remove an ID from the Drop IDs list, select it and click the *Remove* button.

See also: Drag and Drop

1.3.3 The Code Generator

The GUI Designer is integrated with the wED program editor. You can use the wED editor to edit the BASIC source code supporting the GUI project you are working on. Click the Tools menu and select Edit Code or Update Code, or select Edit Code from the popup menu displayed when right-clicking a control in the form or project tree.

When you access the code tools (Edit or Update) from a GUI project that is not yet associated with a code file and item, you can use the integrated *code generator* to create a base program for your project, or you can specify an existing program file and item-ID to associate with your GUI project.

When the *code generator* creates the base program for your project, it uses a "code template" which provides boiler-plate BASIC code segments that are put together to create the program for you. When you first generate the base program for your project, you need to select the appropriate code template

for your application. Built-in templates are provided for *standalone*, *dialog box*, *subroutine*, and *MDI subroutine* code models.

Note that the *subroutine* template is for use in SDI (single document interface) applications, where each form is shown on the desktop. MDI (multiple document interface) forms are shown inside the MDI workspace window.

Code Editor
If you select *Code Editor* from the main menu (or *Edit Code* from the popup menu), AccuTerm will open a wED editor window containing your code.

Update Code
If you select *Update Code* from the main menu, AccuTerm will display the *Update Code* dialog. This dialog is used to check the status of required code sections (initialization, event loop, event decoder, event handler, error handler) in the associated code item. You can use this dialog to automatically add any missing code sections to your program, rebuild the event decoder, and remove obsolete event handlers from the program.

The event decoder is a local subroutine that the code generator creates for you. This subroutine consists of nested CASE constructs which "crack" the event source and dispatch the event to a local event handler subroutine. Do not modify code in the event decoder subroutine - your code may be lost when the decoder is rebuilt!

1.3.4 Code Models

When the *code generator* creates the base program for your GUI project, it uses a "code template" which provides boiler-plate BASIC code segments that are put together to create the program for you. When you first generate the base program for your project, you need to select the appropriate code template for your application. Built-in templates are provided for *standalone*, *dialog box*, *subroutine*, and *MDI subroutine* code models. You can also create your own custom code templates.

Standalone GUI Program
If you select the "standalone" code template, the code generator will create a standalone BASIC program that will load your GUI project, display its main form, dispatch events to event handlers, and create stub event handlers for any events you have specified for the GUI objects in your project. When the last form is closed, the GUI application and program both terminate.

The standalone GUI program is the simplest GUI program model you can use. It is sufficient for very small GUI applications, however there are some severe limitations. First, you must hide the GUI application before you EXECUTE any other program which requires user input, either GUI or character-based. This is because if a GUI form is visible to the user, they will expect to be able to interact with it (click buttons, enter data, etc.). However, since the program that supports the GUI application is no longer in control (it is EXECUTEing another program), the standalone GUI program will not be able to respond to any events.

GUI Dialog Box Subroutine
If you select the "dialog box" code template, the code generator will create a subroutine that displays a GUI dialog box. The dialog box is "modal", meaning that the user must dismiss it by clicking OK or Cancel before the calling program can proceed. There is no problem nesting dialog box subroutines, since each is "modal" with respect to the calling program. You can customize the dialog box subroutine argument list to pass variables containing any initial state data, and return final state data to the calling

program.

Due to its "modal" nature, a dialog box subroutine can be called from any GUI or character-based program. However, a dialog box subroutine should not EXECUTE a standalone GUI program or a character-based program that requires user input, for the same reasons as stated for standalone programs.

Multi-Application Subroutine
The GUI runtime environment supports multiple GUI applications being loaded simultaneously. In order to support this, large applications are partitioned into smaller sub-applications, which are created as separate GUI projects. When using this model (see Multi-Application Model topic later for more information), select "subroutine" as the code template. When you select "subroutine", a callable subroutine will be generated. The subroutine must be called by a "main" program patterned after the included ATGUI.MAIN.SAMPLE program.

Multi-Application MDI Subroutine
The "MDI subroutine" code template uses the same Multi-Application Model mentioned above. This template is only suitable for MDI applications. Using the Multi-Application Model with MDI applications lets all MDI applications share a common main MDI window.

1.3.4.1 Multiple Application Model

AccuTerm 7 GUI supports a multiple application model which can be used when building very large applications. It is not practical to create a monolithic very-large application using the standalone GUI program model due to Windows resource limitations and performance. However, a very-large application can be partitioned into smaller sub-applications, and one or more sub-applications can be run concurrently.

In order to run more than one sub-application at the same time, a master application is used as an event dispatcher, passing events to the correct sub-application. When a sub-application's internal event loop receives an event which does not belong to it, the sub-application simply RETURNs and the master application dispatches the event to the correct sub-application.

Using this model, a very large application is partitioned into smaller sub-applications. Each sub-application is actually a complete GUI application, but its corresponding BASIC code is implemented as a subroutine, rather than as a standalone program.

In order to avoid stressing Windows and the GUI runtime, you can manage which sub-applications are loaded at any given moment. Several custom events are used by the sub-application subroutines to activate the sub-application, load and unload the sub-application, and show and hide forms within the sub-application. The code generator automatically creates these custom events, which are passed as a string literal to ATGUIPOSTEVENT: 'ACTIVATE', 'LOAD', 'UNLOAD', 'SHOW', and 'HIDE'. To start the first sub-application, the master application can use the ATGUIPOSTEVENT to post an 'ACTIVATE' event for the first sub-application. Then, the standard event loop will dispatch the event to the correct sub-application. All custom events are handled in the "Custom Events" section of subroutines created by code generator in the GUI Designer. The 'ACTIVATE' pseudo-event ensures that the application is loaded, and that the first form in the application is visible. If the application is already loaded, the previously active form is activated.

Note: there is no restriction on adding your own custom events; just that the event code you choose must not be numeric.

There are two styles available for multi-application subroutines. If all the sub-applications are SDI select the 'subroutine' code template. On the other hand, if your application uses several MDI sub-applications, select the 'mdisub' code template.

When you load multiple MDI sub-applications, all the sub-applications share a common MDI main window. The main window is created by the first MDI sub-application that is loaded, becoming the "master MDI application". The master MDI application remains in existence until all other MDI sub-applications are unloaded. The master MDI application supplies the default MDI window menu and toolbar. MDI menu and toolbar objects from the other sub-applications are ignored. The default MDI Close event handler, along with the default handler for the 'MDICLOSE' custom event, provide a mechanism for unloading the other sub-applications before the master MDI application. Each sub-application is given a chance to veto the unloading of the entire set of MDI sub-applications by RETURNing from the MDICLOSE custom event before the default handler is executed. If any sub-application RETURNs before the default event code runs, none of the sub-applications will be unloaded.

For SDI sub-applications, when a form receives a Close event, the default Form Close event handler hides the form. Then, if all forms belonging to the sub-application are hidden, the sub-application is unloaded. When the last sub-application is unloaded, the master event loop will receive a Quit event, which terminates the entire application.

For MDI sub-applications, the default Form Close event handler simply hides the form.

1.3.4.2 Custom Code Templates

The code generator in the AccuTerm GUI Designer supports customized code templates. By default, code templates are found in the ...\Program Files\Atwin70\GUILIB folder, although you can change the default location in the GUI Designer Preferences. Please make a copy of one of the existing templates to use as a base for your custom template, then using any text editor, customize the various code sections to meet your development standards.

Code templates are identified by a unique Style attribute, in the [Info] section of the template file. Styles 0 to 49 are reserved for use by Zumasys; user-defined styles should range from 50 to 99. The style selected when code is first generated for a GUI project is stored in the GUI project, so that the code generator can properly update the code at a later time.

The *Type* attribute for code templates should always be "template". The *Name* attribute is shown in the code Code Template dropdown list; the *Description* attribute of the selected template is displayed below the Code Template dropdown.

The code sections are found in the *[Code]* section of the template file. Each section is identified by a numeric "tag" at the beginning of each section. Please do not modify the tags, as the code generator may not function properly.

The code generator performs substitution for several variables found in the code template sections. Besides the variables present in the standard templates, the following variable are available for use by the developer in creating custom code templates:

%%COPYRIGHT%%	the value of the App object Copyright property.
%%AUTHOR%%	the value of the App object Author property.
%%DESC%%	the value of the App object Description property.
%%VERSION%%	the value of the App object Version property.

%%DATE%%	the current date.

1.3.5 Validate & Update Code

Open the *Update Code* tool by using **Tools ► Update Code** from the main menu to validate the structure of the program code associated with the selected GUI project. The *Update Code* tool checks the program for required code sections and provides tools to insert missing sections or update out of date code. It also provides tabs to insert missing event handlers and remove unused event handlers.

The *Code Sections* tab of the *Update Code* dialog shows you the status of various code sections (initialization, event loop, event decoder, event handler, error handler) of the support code for your GUI project. If any of the *Update* buttons are enabled, you can click that button to perform the update indicated.

The first frame lists any standard code sections which are missing and allows you to insert them by clicking the *Update* button.

The middle frame list the status of the code needed to load the first (startup) form. If that code is missing, click the *Update* button to insert it. The generator will assume that the first form in the project tree is the startup form.

The last frame shows the status of the event decoder. If the decoder is out of date, click the *Update* button to rebuild it. The event decoder may become outdated when you add or remove forms or controls from the project, or change settings in the Event Properties tab for any form or control.

Click the Missing Events tab to show any event handlers which are missing.

Click the Unused Events tab to show any event handler in your code which are no longer required.

1.3.6 GUI Form Preview

A *Preview* feature is available to display a preview of the currently selected form in your project. The preview will be displayed by the same AccuTerm component used at runtime, so you can see exactly what your form will look like. Access the preview feature from the *Tools* menu.

A status message is displayed at the top of the preview dialog. Under the status message is a text panel which shows any error messages generated in the preview process. If you want to monitor events in the preview, select the *Show events* check-box. The preview function does not execute any of your event handlers or any code on the host system.

Close the preview form when you are done by clicking the *Cancel* button.

1.3.7 GUI Designer Preferences

The *GUI Designer Preferences* allow you to establish some default options and settings, as well as select the default code templates for use with the code generator. Open the preferences dialog from the GUI Designer Tools -> Settings menu.

Behavior of the Enter key
When you create a new *application* object, the new object's property is initialized with this setting. There are three settings for this option:

> Normal - the Enter key adds lines to *multiline edit controls*, actuates (clicks) an active *command button* or actuates the "OK" button, if one is present.
>
> Same as Tab key - the Enter key works just like the Tab key and activates the next field in the tab order. The "OK" button is not supported.
>
> Same as Tab, except - the Enter key activates the next field, unless a *multiline edit control* or *command button* is active. The "OK" button is not supported.

Auto select text fields
When you create a new *application* object, the new object's property is initialized with this setting.

Disable tree control automatic tooltips
When you create a new *application* object, the new object's property is initialized with this setting.

Style (font and scale)
Select a default font and scale mode in the style box. When you create a new *application* object, the scale mode is initialized with this setting. When you create a new *form* object, the new form's font is initialized with these settings.

Copyright
When you create a new *application* object, the new object's Copyright property is initialized with this setting.

Author
When you create a new *application* object, the new object's Author property is initialized with this setting.

Logo file name
When you create a new *application* object, the new object's Logo property is initialized with this setting.

Code template folder
Enter the path where the code template files are located. The code template files are used by the code generator to create the base program or subroutine for your GUI project. Custom code templates can be added to this folder to suit specific development requirements.

Recent files
Enter the number of items to display in the "Recent Files" list at the bottom of the File menu.

Record locking
Check this check-box if you want the GUI designer to use record locking when reading projects from the host file system.

Designer extensions
Check this check-box if you want to enable GUI designer extensions defined in the host's ACCITERMCTRL file.

1.4 The GUI Runtime

Features in the GUI runtime include creating and initializing *applications*, *forms* and *controls*, deleting *applications*, *forms* and *controls*, setting properties, retrieving property values, calling methods and processing events.

Since the GUI environment is primarily intended to be a front-end for MultiValue data entry programs, each *form* includes a *Reset* method to reset all contained *controls* to their default values. It also includes a *Changed* property which indicates if any contained *controls'* value has been modified (by the user).

Each *form* created must have a unique, user-assigned, ID. Each *control* on a form must also have a unique ID within the scope of its containing *form*. When *controls* are created, the *application* ID, *form* ID and *control* ID must be specified. Additionally, if a *control* is to be created within a "container" *control* (frame, picture or tab), the container *controls's* ID is specified as the parent ID.

Since many MultiValue programs utilize “dots” in variable names; they are legal to use in an ID, however, asterisks are not legal to use in an ID.

The GUI runtime is initialized by the host via the ATGUIINIT2 subroutine. The host sends instructions to the GUI runtime to create GUI objects, which provide the user interface. The user interacts with the GUI objects, and selected events are passed back to the host for handling. Each control documents the set of events that it supports. For more information on event processing, see the "Sequence of Events" topic.

Special GUI functions are included to handle message boxes, input boxes, and common dialogs.

1.4.1 Application

In the AccuTerm GUI environment, elements are arranged in a hierarchy. The *application* is a top-level GUI object. The term *application* refers to a logical collection of forms along with properties which describe how the application as a whole behaves. Application-level properties include how forms are arranged (single or multiple document interface), coordinate scale, description, copyright, author and whether the **ENTER** key acts like the **TAB** key.

1.4.2 Form

In the AccuTerm GUI hierarchy, a *form* is a container which holds user-interface controls. Forms are owned by an application. Depending on its style, a *form* might appear just like other Windows applications, with menus and toolbars, or it might appear as a dialog box, similar to Windows' standard File Open dialog box. Forms may be resizable.

A *form* has properties such as background color, default font and background picture. Controls "inherit" some of their parent *form* properties such as background color and font.

1.4.3 Control

In the AccuTerm GUI hierarchy, a *control* is the lowest element, and is ultimately the element used by users when interacting with your GUI application. Some *controls,* such as frames, pictures, and tab

groups may contain other *controls. Controls* reside on forms and are the building blocks of the graphical interface.

AccuTerm GUI includes a variety of standard *controls* including label, text box (edit), list (including drop-down lists), combo-box, check box, option buttons, command button, grid, frame, picture, index tabs, tree, gauge, timer, menu, toolbar and status bar. Each control type has properties which affect the appearance and functionality of the *control*.

1.4.4 Event sequence

Most user actions trigger only a single event, and no special considerations are necessary when handling these events. However, in some situations, a single user action may trigger multiple events. This topic describes the sequence that these events occur so that you can accommodate this in your application design.

Note: as a general rule, only *user actions* trigger events. Calling most ATGUI... subroutines from the host program do not trigger events. This is by design. There is an exception to this rule: changing the *Visible* property of an *Application* or *Form* may trigger *Activate* and *Deactivate* events for the application or form that is activated or deactivated. This may also trigger *Activate, Deactivate* and *Validate* events for the control that is deactivated on one form and the control that is activated on the other form

DblClick event
A *DblClick* event always follows a *Click* event.

Activate, Deactivate and Validate events
The *Validate* event should occur first, followed by the *Deactivate* event, and finally the *Activate* event. Calling ATGUIACTIVATE while handling any of these events normally cancels subsequent related events. For example, calling ATGUIACTIVATE to re-activate a text-box control, while processing the *Validate* event for that text-box control, should cancel pending *Deactivate* and *Activate* events.

Note that these events are all triggered when activation changes. If a user action does not change the active control, these events do not fire. Examples of actions that do not change the active control include clicking the "X" in the form's control menu, clicking on a menu item, resizing a resizable form or clicking on a control that does not activate (label, picture, frame, gauge, disabled control, etc.). This is not an exhaustive list and other actions might have the same behavior.

Click event
The *Click* event, which often activates a control, is triggered after the *Activate* event. If a control does not support the *Activate* event, the *Click* event occurs, but no *Activate, Deactivate* or *Validate* events will occur.

Close event
The *Close* event is triggered when the user clicks the "X" in a form's control menu (or selects Close from the system menu or clicks a menu item whose ID is *CLOSE). This does not close the form! It only informs the host program that the user intends to close the form. Note that clicking the "X" in the form's control menu does not change the active control, so you will not receive *Validate* or *Deactivate* events prior to the *Close* event. If your *Close* event handler hides or deletes the form, and another form becomes active, the previously active control will receive *Validate* and *Deactivate* events at that time.

Help event
The *Help* event is triggered when the user clicks on the *Help menu item or presses the F1 key for help.

It also occurs when the user clicks a TOPIC hyperlink in the currently displayed help page. This event is fired from the Application object, and requires that GEHELP be included in the Application's EVENTMASK. This event is used for HTML Document help (help type 1).

Quit event
The *Quit* event is triggered when the GUI runtime environment is ready to shut down. This happens when all runtime objects have been deleted.

1.5 Tab Order and the Caption Property

Each form has a “tab order”, which is the order in which controls are activated when the **TAB** key is pressed. The order in which controls are created determines their tab order. The first control created is the first control to receive activation when the form is opened. Certain controls (labels, frames, pictures) cannot be activated, so these are “skipped” when the tab key is pressed, and the next control is activated instead. A control can be removed from the "tab order" by changing its TABSTOP property to zero (or by unchecking the Tab Stop setting in the properties page in the GUI Designer).

The CAPTION property may be used to create keyboard “shortcuts” for certain controls. Activatable controls with a CAPTION property (option group, check box, command button) become activated when their shortcut key is pressed. The shortcut key for a control is specified by inserting an ampersand (&) in the caption text immediately before the desired letter. For example, if the caption text is “Close”, and the desired shortcut key is “C”, then the caption property should be set to “&Close”. To use the keyboard shortcut, the user presses the shortcut key while the **ALT** key is depressed (e.g. **ALT+C**).

Controls with a CAPTION property which cannot be activated (labels and frames) cause the next activatable control (following the label or frame in the tab order) to be activated when the shortcut key is pressed.

To adjust the "tab order" in the GUI Designer, you can drag and drop a control up or down in the project tree. You cannot move a control to another parent in the tree by dragging. If you need to move a control to a different parent, you can use cut and paste.

1.6 Internationalization

AccuTerm GUI allows the text for most runtime messages and prompts to be customized. Each language has its own customization. The default language is English (EN). Languages are identified by their 2 character ISO639 language ID. Custom messages and prompts are stored in the atguisvr.ini file, in the ...\Program Files\Atwin70 folder. A sample atguisvr.ini file is provided in the ...\Program Files\Atwin70\Samples\I18N folder. Two versions of the sample are included: one in standard ANSI character encoding, the other in Unicode. Use the Unicode version for languages that do not use code page 1252.

If you create a translation that you believe would be useful to other developers, and are willing to share your translation, please send it to Zumasys, attn: AccuTerm Development, for distribution with future releases of AccuTerm.

Index

- . -

- A -

- B -

- C -

- D -

- E -

- F -

- G -

- H -

- I -

- L -

- M -

- O -

- P -

- Q -

- R -

- S -

- T -

- U -

- V -