Revision 5.3c **June 18, 2007**

AccuTerm 2K2

Programmers Guide

MultiValue Edition

ACCUSOFT ENTERPRISES
8041 Foothill Blvd.
Sunland, California 91040

WWW.ASENT.COM
Phone (818) 951-1891
FAX (818) 951-3606

Contents

Introduction 1
Scripting 3
AccuTerm Object Reference 13
Customizing the Menu and Toolbar.....71
AccuTerm Programming 79
Wyse Programming 89
ADDS Programming.....107
ANSI Programming.....113
Pick PC Console Programming 137
MultiValue Server Object.....143
GUI Development Environment.....153
Object Bridge.....275
Mouse Support.....281
Appendix A – Wyse Tables.....285
Appendix B – Viewpoint Tables 291
Appendix C – ANSI Tables.....297
Appendix D – ASCII Codes 303
Appendix E – Custom Features 307

INTRODUCTION

This Programming Reference manual is designed to provide developers (and power users) technical information about AccuTerm 2K2. Information on the programming codes for all of the terminals which are emulated by AccuTerm 2K2 are included, as well as AccuTerm's own private programming codes. Also included is a complete reference to AccuTerm's object model, which is necessary when automating AccuTerm either from its own VBA scripting environment, or from another Windows application. Information about AccuTerm's MultiValue host / PC integration features is included as well as a complete guide to AccuTerm GUI which allows you to build GUI front-ends for your MultiValue applications.

SCRIPTING

AccuTerm includes a powerful scripting language similar to the popular Microsoft Visual Basic Programming System, Applications Edition (VBA). The language has been enhanced to allow your script to control almost every aspect of AccuTerm's operation.

This section describes how to create and use scripts, as well as language extensions. The next section describes AccuTerm's automation interface (object model), which is available to scripts and offers complete control over AccuTerm's operation. For general language details, select the Help menu item from the Script dialog; printed documentation is also available from AccuSoft. *This manual does not contain the VBA language reference.*

Creating a Script

Scripts are created by entering VBA statements into the script code window. To activate the script window, pull down the **Tools** menu from the main menu, then select the **Script** menu item.

Scripts usually begin with `Sub Main()` and end with `End Sub`. Enter any legal VBA statements between the `Sub Main()` and `End Sub` statements.

You can create other subroutines and functions which are called by your `Main()` subroutine (or from function keys, popup menus, host commands, etc.) Subroutines begin with the `Sub` keyword followed by the name of the subroutine. If the subroutine requires arguments, enclose them in parentheses. The last statement in a subroutine is `End Sub`.

Functions are similar to subroutines, and begin with the `Function` keyword, and end with the `End Function` statement. Functions return a value; simply assign the return value to the function name within your function code.

Certain declarations must be placed before the `Main()` subroutine. Declare DLL functions (using `Declare Function` or `Declare Sub`), and user-defined data types (using `Begin Type` and `End Type`) before `Sub Main()`. Within a subroutine or function, declare any local variables using the `Dim` or `Static` statement.

In addition to local references to subroutines and functions defined in your script, those subroutines and functions may also be called from outside of your script in three ways: First, they may be called by script commands sent by the host (see **AccuTerm Programming**). Second they may be called by script commands programmed into function keys (see **Keyboard Settings**). Third, they may be called by script commands executed in response to custom menu and toolbar actions (see **Customizing the Menu and Toolbar**).

Saving a Script File
After the script has been created (or modified), save the script by clicking the  button or pull down the **File** menu (from the script window), and select the **Save** or **Save As** option.

Loading a Script File
To load a script file, click the  button or select the **Open** option from the **File** menu.

Loading and Running a Script File
To load and run a script file, pull down the **File** menu and select **Run**. Enter or select the name of the script file to run, and click the **OK** button. *Note: execution always begins with* `Sub Main().`

Closing the Script Window
Select the **Close** item from the **File** menu to close the script window. If there are any unsaved changes to the current script, you will be prompted to save changes.

Printing a Script
Select the **Print** item from the **File** menu to print the current script. Select **Print Setup** to select the printer used to print the script.

Editing Scripts
The **Edit** menu provides many editing functions such as **Undo**, **Redo**, **Cut**, **Copy**, **Paste**, **Delete**, **Select all**, **Indent**, **Outdent**, **Find** and **Replace**. The

font used in the script window may be selected by choosing the **Font** item, and the dialog editor may be invoked by choosing the **User Dialog** item.

Running a Script

To run a script from the script window, create a new script or load an existing script into the script window. Pull down the **Run** menu and select **Start (Resume)** or click the  button. To single step the script, see the topic on **Debugging a Script**. To terminate the script, select **End** from the **Run** menu or click on the  button. To suspend execution, select **Pause** from the **Run** menu or click the  button. To resume execution, select **Start (Resume)** or click the  button.

Running a Script from the Command Line

To run a script from the Atwin2k2.exe command line, append the name of the script file to the shortcut target or command line. When AccuTerm starts, it loads the script into a hidden script window, and executes its `Main` subroutine. You can use a command line script to open up a number of sessions, the log a user on, etc. You can also create a shortcut to an AccuTerm 2K2 script file (.atsc extension). When you open a shortcut to a script file, AccuTerm will open and the script file will be executed.

Running a Script from a Layout File

You can run a script automatically when opening a *layout* file. To save a script with a layout file, open all sessions to be included in the layout file. Open the script window, then load the script you want to automatically run. Return to the main AccuTerm window, and select **File Save Layout**. When the layout file is opened next, the script file will be loaded and its Sub Main() will be executed. *Note: when opening a layout file, all sessions are opened before executing the script.*

Running a Script from a Function Key

To run a script from a function key, simply program the function key (see **Keyboard Settings**) with script statements enclosed in brackets **[ ]**. Multiple statements, up to a maximum of 250 characters, may be programmed in a function key. Often, the programmed key would simply call an external script file using the `Chain` statement. *Be sure to follow the normal function key syntax which requires using two backslash(\\ ) or caret (^^) characters when using either of these characters in the script.*

Running a Script from a Menu or Toolbar

Running a script from a menu item or toolbar button is similar to running a script from a function key. In the **Menu Designer** (select **Customize Menu** from the **Tools** menu in the main menu), set the **Action** property of the menu or toolbar item to the desired script statements enclosed in brackets **[ ]**. The same syntax rules that apply to function key script also apply to menu and toolbar scripts.

Running a Script from the Host Computer

To run a script from the host system, send the private AccuTerm command:

ESC STX **P** ***script CR***

where ***script*** is the text of a script to execute. Each script statement is separated by a ***LF*** or ***EM*** control character, and the entire script is terminated with a ***CR*** (carriage return character). Often, a **Chain** statement, which executes another script file, is the entire script sent from the host. Also, when a Main script is loaded in the script window, the script command sent by the host might simply be the name of a subroutine to call (possibly with arguments).

Debugging a Script

The script window contains menu items and buttons for single stepping, interrupting and resuming script execution, and setting breakpoints and watch expressions. Use **Debug Step Into** () to execute the next statement; **Debug Step Over** () to execute the next statement, subroutine or function call. Select **Run Pause** or click  to interrupt execution; select **Run Start (Resume)** or click  to resume execution. To set or clear a breakpoint, click on the desired script line, then select **Debug Toggle Breakpoint** or click the  button. Lines containing breakpoints are marked with a large dot in the left margin. To add a watch expression, position the cursor on the desired expression, then select **Debug Add Watch** or click the  button. To remove a watch expression, click on the **Watch** tab, select the expression you want to delete, and press the **Del** key.

Controlling AccuTerm with Scripts

From a script's perspective, AccuTerm consists of a set of objects. These objects have properties whose values can be examined and altered, and methods which can be invoked. The main object, `AccuTerm`, may be used to control the general settings of the AccuTerm application - those items which are set in the **General Settings** tab of the **Settings** dialog box. The `Session` object is the most useful object for controlling AccuTerm. Using the `Session` object, you can change any of the session **Settings**, communicate with the host system, manipulate the screen, etc. The following chapter, **AccuTerm Object Reference**, describes AccuTerm's automation interface, or *object model*.

There are several built-in objects which can be used in scripts running in AccuTerm. These are the `AccuTerm` object, the `Sessions` collection, the `ActiveSession` object, and the `InitSession` object.

`AccuTerm` object

The `AccuTerm` object is the top-level application object, and is described in detail in the next chapter.

`Sessions` collection

The `Sessions` collection is a collection of `Session` objects. There is one `Session` object in the collection for each currently open session. The first session in the collection is `Sessions(0)`. `Sessions` is simply a shortcut for `AccuTerm.Sessions`. The `Session` object type is described in detail in the next chapter.

`ActiveSession` object

The `ActiveSession` object is an object of type `Session`, which refers to the currently active session. The currently active session is the session whose title bar is highlighted.

`InitSession` object

The `InitSession` object is an object of type `Session`, which refers to the session which initiated execution of the current script. This usually refers to the same session as `ActiveSession`. However, a non-active session could run a script if the host initiates execution using an escape sequence; in this case the two objects refer to different sessions.

Script Language Extensions

Several statements and functions have been added to the VBA language used by AccuTerm. These extensions implement features which may be useful in AccuTerm scripts.

`AppActivate` statement

```
AppActivate title$
```

Activates the application identified by ***title$***. If no window exists with ***title$***, the first window with a title that begins with ***title$*** is activated. If no window matches, an error occurs.

`AppClose` statement

```
AppClose [title$]
```

Closes the named application. The ***title$*** parameter is a String containing the name of the application. If the ***title$*** parameter is absent, then the AppClose statement closes the active application.

`AppFind` function

```
AppFind( title$ )
```

Returns a String containing the full name of the application matching the partial ***title$***.

`AppGetActive` function

```
AppGetActive()
```

Returns a String containing the name of the active application.

`AppGetPosition` statement

```
AppGetPosition X, Y, width, height [, title$]
```

Retrieves the position of the named application. ***X***, ***Y***, ***width***, ***height*** are integer variables into which the application's position and size are stored. ***Title$*** is the name of the application whose size is being retrieved. If ***title$*** is not specified, then the currently active application is assumed.

`AppGetState` function

```
AppGetState([title$])
```

Returns an Integer specifying the state of the top-level window.

`AppHide` statement

```
AppHide [title$]
```

Hides the named application.

`AppList` statement

`AppList` ***AppNames$()***

Fills an array with the names of all open applications. The ***AppNames$*** parameter must specify either a zero- or one-dimensioned dynamic String array. The array will be redimensioned to match the number of open applications. After calling this function, you can use LBound and UBound to determine the new size of the array.

`AppMaximize` statement

`AppMaximize` [***title$***]

Maximizes the named application.

`AppMinimize` statement

`AppMinimize` [***title$***]

Minimizes the named application.

`AppMove` statement

`AppMove` ***X***, ***Y*** [, ***title$***]

Sets the upper left corner of the named application to a given location.

`AppRestore` statement

`AppRestore` [***title$***]

Restores the named application.

`AppSetState` statement

`AppSetState` ***newstate*** [, ***title$***]

Maximizes, minimizes, or restores the named application, depending on the value of ***newstate***. The application is maximized if ***newstate*** is 1, minimized if ***newstate*** is 2 and restored if ***newstate*** is 3.

`AppShow` statement

`AppShow` [***title$***]

Shows the named application.

`AppSize` statement

`AppSize` ***width***, ***height*** [, ***title$***]

Sets the size of the named application.

`Chain` statement

`Chain` ***filename*** [***arguments***]

`Chain` ***filename*** | ***macroname*** [***arguments***]

This statement transfers control to the specified script file. ***Filename*** is a string expression, and may contain a complete path name. If ***macroname*** is present, it must be separated from ***filename*** by a vertical bar (|) and execution begins with subroutine ***macroname***. Otherwise, execution begins with `Sub Main()` in the new script file. Optional ***arguments*** may be appended to the filename string separated by a space character. The arguments may be retrieved in the chained-to script by using the `Command()` function. Execution never returns to the calling script. *Note: both* ***filename*** *and* ***macroname*** *and optional* ***arguments*** *are contained in a single string expression, such as*

"C:\ATWIN\SCRIPTS\MYSCRIPT.ATSC|FOO IMAGE XX.JPG"

`Common` collection

`Common.Clear`

`Common(`***key***`) = ` ***value***

variable `= Common(`***key***`)`

The `Common` collection is used to store values which may be saved between script executions, or shared with between several running scripts. That is, one session can set a value in the `Common` collection, which another session can retrieve it. Items stored in the `Common` collection are referenced by "keys". The ***key*** is a string argument. The values stored in the `Common` collection are *variants*, and thus can contain any data type. The `Common` collection can be reset by using the `Clear` method.

`Debug.Print` statement

`Debug.Print` ***text***

This statement prints the specified ***text*** to a special debugging window. Use **Window Debug** menu to show the debugging window.

`FileExists` function

`FileExists(` ***name$*** `)`

Returns True if file exists, otherwise returns False.

`Item` function

`Item(` ***text$***, ***first***, ***last***, [***delim$***] `)`

Returns all the items between ***first*** and ***last*** within the specified formatted text list. Items are substrings of a delimited text string. Items, by default, are

separated by commas and/or end-of-lines. This can be changed by specifying different delimiters in the ***delim$*** parameter.

`ItemCount` function

`ItemCount(` ***text$***, [***delim$***])

Returns an Integer containing the number of items in the specified delimited text. Items are substrings of a delimited text string. Items, by default, are separated by commas and/or end-of-lines. This can be changed by specifying different delimiters in the ***delim$*** parameter.

`Line` function

`Line(` ***text$***, ***first***, ***last***)

Returns a String containing a single line or a group of lines between ***first*** and ***last***.

`LineCount` function

`LineCount(`***text$***)

Returns an Integer representing the number of lines in ***text$***.

`OpenFileName` function

`OpenFileName(` [***title$*** [, ***extension$***]])

Displays a dialog box that prompts the user to select from a list of files, returning the full pathname of the file the user selects or a zero-length string if the user selects Cancel. If ***title$*** is specified, then it is used as the title of the dialog box, otherwise, "Open" is used. If ***extension$*** is specified, then it specifies a list of file types and extensions - *type:ext[,ext][;type:ext[,ext]]…* where *type* is a description of the type of file, and *ext* is the extension pattern. For example, to display "All Files" and "Documents", the ***extension$*** argument might be:

```
"All files:*.*;Documents:*.txt,*.doc"
```

`Pause` statement

`Pause` ***seconds***

This statement causes the script to pause for the specified number of seconds. While the script is paused, normal terminal functions are operational, including any file transfers in progress.

`Random` function

`Random(`***min, max***)

Returns a Long value greater than or equal to ***min*** and less than or equal to ***max***.

`SaveFileName` function

`SaveFileName` ([***title$*** [, ***extension$***]])

Displays a dialog box that prompts the user to select from a list of files, returning the full pathname of the file the user selects or a zero-length string if the user selects Cancel. If ***title$*** is specified, then it is used as the title of the dialog box, otherwise, "Save As" is used. If ***extension$*** is specified, then it specifies a list of file types (see `OpenFileName` function for details).

`Sleep` statement

`Sleep` ***milliseconds***

This statement causes the script to pause for the specified number of milliseconds. While the script is paused, normal terminal functions are operational, including any file transfers in progress.

`Word` function

`Word(` ***text$***, ***first***, ***last*** `)`

Returns a String containing a single word or a group of words between ***first*** and ***last***.

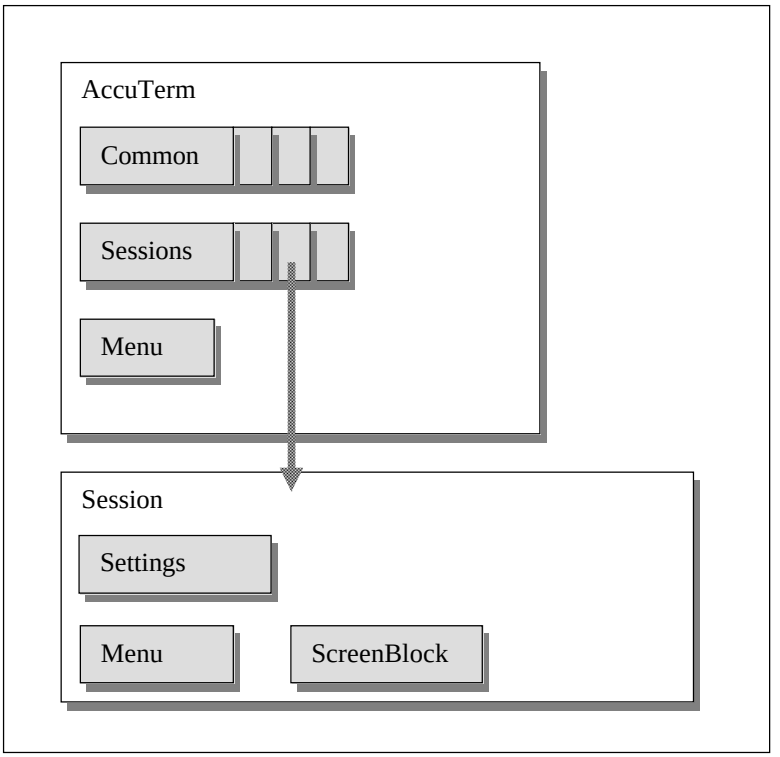
`WordCount` function

`WordCount(` ***text$*** `)`

Returns an Integer representing the number of words in ***text$***. Words are separated by spaces, tabs, and end-of-lines.

ACCUTERM OBJECT REFERENCE

AccuTerm 2K2 exposes a rich automation interface (object model). This allows AccuTerm to act as an automation server to any number of client applications. AccuTerm's objects are arranged in a hierarchy, with the `AccuTerm` application object at the top. The object hierarchy is shown in diagram below.



Properties
This following sections describe the properties, methods and events available from AccuTerm objects. To reference a property, simply append a period (.) followed by the property name to an object variable of the appropriate type. To get the value of a property, simply assign the property reference to a variable of the appropriate type; to set the value, assign an appropriate expression, constant or variable to the property reference.

Methods
To reference a method, simply append a period (.) followed by the method name to an object variable of the appropriate type. Some methods return a value; for these methods, assign the result to a variable of the appropriate type. Some methods accept one or more arguments separated by commas. If the method also returns a value, enclose the argument list in parentheses. Optional arguments are shown enclosed in brackets [ ]; the brackets are not part of the syntax.

Events
Certain of AccuTerm's objects have the ability to fire events. To use events, the client application must be capable of responding to events. For example, to use the `Session` object's `DataReady` event, the following declaration is required in a VB form or class:

```
Dim WithEvents objSession As
AccuTermClasses.Session
```

Type Library
AccuTerm 2K2 includes a type library, ATWIN2K2.TLB, which contains information about AccuTerm's public objects, including their properties, methods, events, arguments and constants. When specifying the object type in a declaration, always use the type library as a reference. This will ensure compatibility with future versions of AccuTerm. For example, to declare a VB object variable as type "AccuTerm", add a reference to "AccuTerm 2K2 Classes" (Project->References), then use the following declaration:

```
Dim obj As AccuTermClasses.AccuTerm
```

The AccuTerm Object
The AccuTerm object is AccuTerm's top-level application object. To access the AccuTerm object of a running instance of AccuTerm, use the following syntax:

```
Dim obj as Object
Set obj = GetObject(, "ATWin32.AccuTerm")
```

To create a new instance of AccuTerm, use:

```
Dim obj as Object
Set obj = CreateObject("ATWin32.AccuTerm")
```

The properties and methods of the AccuTerm object are described in the following pages. Use these properties and methods to control the overall application. The AccuTerm object contains the Sessions collection, which has one element for each open session. The ActiveSession property may be used to obtain a reference to the Session object which is currently active.

Note: within AccuTerm's own script environment, you do not need to use GetObject( ) *to obtain a reference to the* AccuTerm *application object. The* AccuTerm *object is a built-in object and can simply be referenced as* AccuTerm.

Activate method
Makes AccuTerm the active application.

ActiveSession property (Session object)
The ActiveSession object is an object of type Session, which refers to the currently active session. The currently active session is the session whose title bar is highlighted.

Arrange method
AccuTerm.Arrange [***style***]
Arranges AccuTerm session windows. Set ***style*** to 0 for cascade, 1 for tile horizontal, 2 for tile vertical and 3 to arrange icons.

AutoClose property (boolean)
If non-zero, then when the last open session is closed, AccuTerm will terminate.

`Close` method

`AccuTerm.Close` [***prompt***]

Closes AccuTerm. If ***prompt*** is True (non-zero) and any changes have been made to session configuration settings, the user will be prompted save the changed settings.

This is the same as the `Terminate` method, but is *not* available when using late binding.

`Common` property (collection object)

Returns a reference to the `Common` collection object. The `Common` collection may be used to save global variables so that they may be shared between different instances of scripts and OLE clients.

`CustomMouseTable` property (string)

This is the file name of the current custom mouse table. See "Mouse Support" chapter for more information on using a custom mouse table.

`FuncBarPos` property (integer)

Set to 0 to display AccuTerm's function key bar at the bottom of the screen or 1 to display the function key bar as the top of the screen.

`FuncBarStyle` property (integer)

Set to 1 to display the function key number as a tiny number in the corner of the button, or 0 to not display the key number.

`FuncBarVisible` property (integer)

Set to True (non-zero) to display AccuTerm's function key bar, otherwise set to False (zero).

`Height` property (long)

This is the application window height in pixels.

`Hide` method

This method makes AccuTerm's main window invisible.

`LargeIcons` property (integer)

Set to True (non-zero) to show large icons in the toolbar or False (zero) to show small icons.

`LayoutFileName` property (string)
This read-only property returns the name of the currently open layout file, if any.

`Left` property (long)
This is the horizontal position of the application window in pixels.

`Menu` property (`Menu` object)
This returns a reference to the application `Menu` object, which may be used to customize AccuTerm's menus and toolbars. For more information on the `Menu` object, refer to chapter titled "Customizing the Menu and Toolbar".

`Move` method
`AccuTerm.Move` ***left*** [, ***top*** [, ***width*** [, ***height***]]]
Repositions and resizes AccuTerm's main window. The positions and dimensions are specified in pixels.

`NoCloseWarning` property (integer)
Set to True (non-zero) to suppress the disconnect warning when closing a session.

`OpenLayout` method
`AccuTerm.OpenLayout` ***filename***
Opens the specified layout file.

`PhoneBookName` property (string)
This is the file name of the Auto-Dial phone directory.

`ProductLicenseType` property (integer)
This read-only property returns the product license type: 1=single user, 2=site or small business license, 3=enterprise license, 5 = developer license, 7 = Internet Edition client, 9=demonstration version.

`ProductName` property (string)
This read-only property returns the string "ACCUTERM/WIN32".

`ProductRelease` property (string)
This read-only property returns the current release of AccuTerm 2K2.

`ProductSerialNumber` property (long)
This read-only property returns the serial number of AccuTerm 2K2.

`RecentListSize` property (integer)
Specifies the number of entries in the recent file list.

`RegisteredCompany` property (string)
This read-only property returns the registered company name.

`RegisteredLocation` property (string)
This read-only property returns the registered location.

`RegisteredUser` property (string)
This read-only property returns the registered user name.

`Resize` method
`AccuTerm.Resize` ***width, height***
Resizes AccuTerm's main window. The dimensions are specified in pixels.

`SaveLayout` method
`AccuTerm.SaveLayout` [***filename***]
Saves the current (or specified) layout file.

`SessionBarVisible` property (integer)
Set to True (non-zero) to display AccuTerm's session quick-bar, otherwise set to False (zero).

`Sessions` property (collection)
Returns a reference to the `Sessions` collection. This collection has one member for each open session. The first session is `Sessions(0)`.

`SettingsDialog` method

`AccuTerm.SettingsDialog [` ***CurTab*** `[, ` ***Tabs*** `]]`

Displays the Settings dialog box. Optionally specify ***CurTab*** (letter of the currently selected tab) and a list of ***Tabs*** to be displayed. The ***Tabs*** list is a string with a single letter for each visible tab. Letters which may be used for ***Tabs*** are:

G = general
P = printer
X = file transfer
D = device (connection)
T = term type
S = screen
F = fonts
C = colors
K = function keys
M = miscellaneous

`Show` method

Makes AccuTerm's main window visible.

`SingleInstance` property (integer)

Set to True (non-zero) to allow only one instance of AccuTerm. To allow multiple instances, set to False (zero).

`StatusLineVisible` property (integer)

Set to True (non-zero) to display AccuTerm's status line, otherwise set to False (zero).

`TelnetHostsFile` property (string)

This is the file name of the hosts file. The hosts file is used to list available hosts when setting up a telnet session.

`Terminate` method

`AccuTerm.Terminate [`***prompt***]

Closes AccuTerm. If ***prompt*** is True (non-zero) and any changes have been made to session configuration settings, the user will be prompted save the changed settings

This is the same as the `Close` method, and is available when using late binding.

`TitleFormat` property (integer)
Selects the format of the session title:

0	caption (ID)
1	caption [ID]
2	(ID) caption
3	[ID] caption

`ToolbarVisible` property (integer)
Set to True (non-zero) to display AccuTerm's tool bar, otherwise set to False (zero).

`Top` property (long)
This is the vertical position of the application window in pixels.

`TrackKeyboardState` property (integer)
Set to True (non-zero) to track the state of the **CapsLock** and **ScrollLock** keys. When tracking is enabled, the state of these keys is saved and restored when the user switches to and from the AccuTerm application.

`Visible` property (boolean)
Set to True (non-zero) to make AccuTerm's main window visible. Set to False to hide AccuTerm's main window.

`Width` property (long)
This is the application window width in pixels.

`WindowState` property (integer)
This is the state of AccuTerm's main window. Set to 0 for normal, 1 for minimized or 2 for maximized.

The Sessions Collection

You can create (open) a new session, and you can access any opened session by using the `Sessions` collection. The `Sessions` collection acts like an array of `Session` objects with one element for each open session.

The `Sessions` collection is used to access any open session by using an array reference:

`Sessions(`***index***`)`

returns a `Session` object for session ***index***. Session indexes are numbered from zero; the first open session is `Sessions(0)`. The last session is `Sessions(Sessions.Count() - 1)`.

`Sessions.Add` method

`set` ***object*** `= Sessions.Add(`[***filename*** [`,` ***state*** [`,` ***mode*** [`,` ***hidden***]]]]`)`

The `Sessions.Add` method creates and initializes a new session. The return value is a `Session` object: All arguments are optional:

filename	configuration file name used to initialize session.
state	initial window state (0 = normal, 1 = minimized, 2 = maximized)
mode	input mode (0 = normal, 1 = synchronous, 2 = disconnected)
hidden	0 if new session is initially visible, 1 if hidden
object	session object created by this method

`Sessions.Count` method

count `= Sessions.Count()`

The `Sessions.Count` method returns the number of open sessions.

The Session Object

When you use AccuTerm, you create "terminal sessions" to communicate with a host computer. AccuTerm's multiple document interface (MDI) allows you to create as many sessions as you desire, all connected to different host systems (or more than one session connected to a single host). AccuTerm's `Session` object gives you access to session properties such as port, font, screen colors, etc.) and allows you to invoke session methods (clear, delete, input, output, etc.)

Predefined Session Objects
The currently active session may be accessed using the `ActiveSession` object. When a script is invoked from a function key, menu, toolbar button, or from the host computer system, you can access the session which invoked the script by referencing the `InitSession` object. *The* `InitSession` *object is only valid in the context of a script running in the AccuTerm application.*

Creating a New Session
There are two ways to create a new session. First, you can declare a `Session` object variable, and use the `New` keyword with the `Set` statement to create the new session:

```
Dim S as Session
Set S = New Session
```

These statements will create a new session. All session properties will be set to their default values. You access the properties and methods of the session using the `Session` object variable `S`.

An alternate way of creating a new session is to use the `Add` method of the `Sessions` collection object as described above.

Session Object Reference

`Activate` method
Makes the session the "active" session. There is no return value.

`Ansi8Bit` property (boolean)
If this setting is non-zero, AccuTerm sends 8 bit control codes. Othewise the eqivalent 7-bit escape sequence is sent. This property is only effective when AccuTerm is emulating one of the VT terminals.

`AnsiAppCursor` property (boolean)
If this setting is non-zero, AccuTerm sends "application codes" instead of "cursor codes" when cursor keys are pressed. This property is only effective when AccuTerm is emulating one of the VT terminals.

`AnsiAppKeypad` property (boolean)
If this setting is non-zero, AccuTerm sends "application codes" instead of numeric characters when key on the numeric keypad are pressed. This

property is only effective when AccuTerm is emulating one of the VT terminals.

`AnsiAutoPrint` property (boolean)
When this setting is True (non-zero), the "auto print" slave printer function works the same as a real VT terminal. That is, when AccuTerm is in "auto print" mode, the entire screen line that the cursor is on is printed when AccuTerm receives a ***CR***, ***LF*** or ***FF*** control code. If this option is False (zero), then text is printed as it is received from the host (similar to Wyse auto-print mode).

`Answerback` property (string)
This is the string that AccuTerm returns to the host system when the host requests the "answerback message".

`AsciiDelay` property (integer)
Specifies the interline delay used for ASCII file uploads in milliseconds.

`AsciiEOL` property (integer)
Specifies the line ending sequence for ASCII file uploads. Set to 0 for ***CR***, 1 for ***LF*** or 2 for ***CR***+***LF***.

`AutoAnswer` property (boolean)
If this setting is non-zero, AccuTerm will answer incoming calls when the session is connected to a modem.

`AutoClose` property (boolean)
If this setting is non-zero, AccuTerm will close the session when it becomes "disconnected".(modem disconnects or host system drops network connection).

`Baud` property (integer)
This is the baud rate used by the serial port attached to the session. This property is only meaningful when the communication device type is "Serial Port". Acceptable baud rates are `atBaud300`, `atBaud1200`, `atBaud2400`, `atBaud9600`, `atBaud14400`, `atBaud19200`, `atBaud38400`, `atBaud57600` and `atBaud115200`.

`BkspSendsDel` property (boolean)
If this setting is True (non-zero), pressing the Backspace key causes AccuTerm to send the ***DEL*** control code. Otherwise, AccuTerm sends the ***BS*** control code.

`BoldFont` property (boolean)
Set to True (non-zero) if terminal font is bold, False (zero) if font is normal.

`Break` method
Sends a "break" signal to the host connected to the session. There is no return value.

`BytesIn` property (long)
This read-only property returns the number of bytes received from the host since the session was opened.

`BytesOut` property (long)
This read-only property returns the number of bytes transmitted to the host since the session was opened.

`CapsLock` property (boolean)
Set to True (non-zero) to set the CapsLock keyboard state, False (zero) if CapsLock keyboard state is reset.

`Caption` property (string)
Set or returns the session caption (title).

`Capture` method

session*.`Capture` *filename*, *source*, *mode

Initiates or terminates data capture mode. If ***source*** is zero, capture mode for session is terminated. Set ***source*** to 1 to initiate capture of received data, 2 to initiate capture of printed data. Set ***mode*** to 0 to create a new file, 1 to overwrite an existing file, 2 to append to existing file, or 3 to capture to the clipboard instead of a file. Add 128 to ***mode*** to filter out control characters from the captured data. If ***source*** is zero or ***mode*** is 3 or 131, then ***filename*** is ignored, otherwise it is the name of the file where captured data is saved.

`CaptureEnd` method
Terminates any capture operation and closes the capture file.

`CaptureFileName` property (string)
The name of the file name used in the last capture operation. This property is read-only.

`CaptureMode` property (integer)
The current capture mode (0 to create a new file, 1 to overwrite an existing file, 2 to append to existing file, or 3 to capture to the clipboard instead of a file). If capture mode has 128 added to it, this indicates that control characters are being filtered from the captured data. This property is read-only.

`CaptureSource` property (integer)
The current capture source (1 to capture received data, 2 to capture printed data). This property is read-only.

`Changed` property (integer)
Non-zero if the settings for the session have been modified. Changing this property to zero prevents the warning about saving changes when the session is closed.

`Charset` property (string)
The character set used by the host. The only valid values are NULL for the default character set for the selected terminal emulation, or "NATIVE" if the host uses the current 8-bit (non-Unicode) Windows character set.

`Clear` method

session`.Clear [`***left*** `[,` ***top*** `[,` ***right*** `[,` ***bottom*** `[,` ***color***`]]]]]`

Clears block of text from session screen to specified background color. Default coordinates are the upper left and lower right corners of the screen. To specify ***color***, use the color index described under `Color` property.

`ClearSelection` method
This removes any selection rectangle from the session screen. There is no return value. This method is the same as the `Deselect` method.

`Close` method

session.Close [***prompt***]

This closes the session. If ***prompt*** = 1 or 3, and if any session settings have been modified subsequent to loading or saving the session, the user will be prompted whether to save the settings. If ***prompt*** = 2 or 3, and the session is connected via a network or dialup connection, the user will be prompted to disconnect. This is the same as the `Terminate` method, but is not available when using late binding.

`Col`, `Row` properties (integer)
The current cursor column or row. The leftmost column is zero and the topmost is zero. Also see the `Locate` method.

`Color` property (integer)
This is the foreground/background color index used when text is displayed using the `SetText` method. To determine the color index, select the foreground and background colors from the following table, and add the indices of each to form the correct color index:

Color	**Foreground**	**Background**
Black	0	0
Dark Blue	1	16
Dark Green	2	32
Turquoise	3	48
Dark Red	4	64
Purple	5	80
Olive	6	96
Light Grey	7	112
Dark Grey	8	128
Blue	9	144
Green	10	160
Cyan	11	176
Red	12	192
Magenta	13	208
Yellow	14	224
White	15	240

Note: the actual colors may be different if the palette has been modified!

`Colors()` property (array of integer)
This array maps foreground and background colors (using the color index described above) to visual attribute combinations. The index to the `Colors()` array is the attribute code shown in the following table:

Index	Attribute Description	Index	Attribute Description
0	normal	22	dim reverse blink
1	blank	24	dim underline
2	blink	26	dim underline blink
4	reverse	28	dim underline reverse
5	reverse blank	30	dim underline reverse blink
6	reverse blink	32	bright
8	undereline	34	bright blink
10	underline blink	36	bright reverse
12	underline reverse	38	bright reverse blink
14	underline reverse blink	40	bright underline
16	dim	42	bright underline blink
18	dim blink	44	bright underline reverse
20	dim reverse	46	bright underline reverse blink

The value of this property may include a border effect style which is associated with the visual attribute. The border effect styles are combined with the foreground and background color as follows:

none = fg + bg
inset = fb + bg + 16384
raised = fg + bg - 32768
flat = fg + bg - 16384

`Cols` , `Rows` properties (integer)
The current number of columns or rows. These are read-only properties. Changing the screen mode between normal or extended mode (see `ExtMode`, `NormMode` and `ScrMode` methods), or changing the value of the `ExtCols`, `ExtRows`, `NormCols` or `NormRows` properties may affect the `Cols` and `Rows` properties.

`Connected` property (boolean)
This read-only property reflects the connection status of the session. For serial connections, it reflects the state of the CD signal; for network connections, it is non-zero of the network connection is OK.

`ConnectTimeout` property (integer)
This is the number of seconds to wait while attempting to connect before a timeout error occurs.

`Copy` method

***session*`.Copy` [*left* [`,` *top* [`,` *right* [`,` *bottom*]]]]**

Copies a block of text from session screen to the clipboard. Default coordinates are the upper left and lower right corners of the screen or the current selection, if one is present. Negative row numbers will copy from the history buffer; row –1 is the last history row, etc.

`CopyHistory` method

***session*`.CopyHistory` [*left* [`,` *top* [`,` *right* [`,` *bottom*]]]]**

Copies a block of text from session history buffer to the clipboard. Default coordinates are the upper left and lower right corners of the history buffer.

`CopyPasteShortcut` property (integer)
Sets to 0 for no shortcut, 1 to use **CTRL** + **INS** / **SHIFT** + **INS**, or 2 to use **CTRL** + **C** / **CTRL** + **V** for Copy / Paste shortcuts.

`CursorType` property (boolean)
Set to True (non-zero) if cursor shown as a block, False (zero) if cursor shown as an underline.

`CustomMouseTable` property (string)
This is the file name of the custom mouse table file for the session. The initial value of this property is the global mouse table file. See **Mouse Support** for more information on using a custom mouse table.

`DataBits` property (integer)
This is the number of data bits used by the serial port attached to the session. This property is only meaningful when the communication device type is `atDevSERIAL` or `atDevMODEM`. Acceptable values are 7 or 8. After changing this property, use the `Reset` method for the change to take effect.

`DataReady` event

***object*`_DataReady()`**

This event is fired when the session's `InputMode` property is non-zero, and data has been received from the host and is ready to be read using the `Input` method.

`DefaultCaptureDir` property (string)
This is the default destination directory used for file capture operations which do not specify a directory.

`DefaultXferDir` property (string)
This is the default destination directory used for file transfer operations which do not specify a directory.

`DefaultXferMode` property (integer)
This is default transfer mode: 0 for text, 1 for binary.

`DefaultXferOverwrite` property (integer)
This is default overwrite setting for received files. Set to True (non-zero) to allow overwrites, else set to False (zero).

`Delete` method
This deletes (closes) the session. There is no return value.

`Deselect` method
This removes any selection rectangle from the session screen. There is no return value.

This method is the same as the `ClearSelection` method.

`Device` property (integer)
This is the communications device type attached to the session. The device type may be `atDevNONE` for no device (disconnects session), `atDevSERIAL`, `atDevPICLAN`, `atDevTELNET`, `atDevSSH` or `atDevMODEM`.

`Dial` method

result = ***session***`.Dial(`***PhoneNumber***`)`

Uses the Auto-Dialer to dial a specified phone number. The return value specifies the result of the dialing operation. See the `DialStatus` property for a description of the return codes. This method only works if the `Device` property is `atDevMODEM`.

`DialStatus` property (integer)
This read-only property reflects the status of the last `Dial` or `HangUp` method. Status codes are described in the following table:

Status	Description
-1	Dialing (or hang-up) in progress.
0	Not connected (successful hang-up).
1	Connected (successful dial).
2	Unable to initialize modem.
3	Requested number is busy.
4	Requested number did not answer.
5	Unable to connect.
6	Unable to hang-up.
7	Modem is in use.
8	Invalid phone number.
9	Billing rejected.
10	No dial tone.

`DisableAppMode` property (boolean)
Set to True (non-zero) to prevent AccuTerm from entering "keypad application mode" or "cursor application mode" when running one of the VT emulations. Set to False (zero) to allow application mode.

`Download` method

result = ***session***.`Download(`***target*** , ***protocol*** , ***binary*** [, ***overwrite***]`)`

Downloads a file from a host system to the user's PC. ***Target*** is the name of the destination file (ASCII or XModem protocol), or the name of the destination directory (Kermit, YModem or ZModem protocol). ***Protocol*** is `atProtocolASCII, atProtocolKermit, atProtocolXmodem, atProtocolYmodem` or `atProtocolZmodem`. ***Binary*** is True (non-zero) for binary transfer mode, False (zero) for text transfer mode. ***Overwrite*** must be `atProtect` or `atOverwrite`, and is only meaningful for ASCII and Xmodem downloads.

`Duplex` property (integer)
This property sets the communications duplex mode to local (no communication to or from host: `atDuplexLOCAL`), full (remote echo: `atDuplexFULL`) or half (local echo: `atDuplexHALF`).

`Emulate` method

session*.`Emulate` *text

Causes the terminal emulator to process the specified text, including control codes and escape sequences.

`ExtCols`, `ExtRows` properties (integer)

These properties define the terminal screen dimensions when the terminal is placed into "extended mode", also known as "132 column mode".

`Extension` event

***object*_`Extension(`*text* `as String)`

This event is fired when one of the extension prefix and suffix characters have been received by the emulation engine. This event is used to extend the functionality of the emulation engine. Use the `SetExtension` method to specify the prefix and suffix characters.

`ExtMode` method

Causes the session to switch to extended (132 column) mode.

`FileName` property (string)

This is the name of the configuration file opened by the session. If there is no configuration file, then this property is NULL. If you assign a new name to this property, the session will be re-initialized with the settings from the new file.

`FKeys()` property (array of string)

This array contains the programmed function and editing key strings. The index is formed by combining (adding) the modifier value with the virtual key number. One of the special modifiers for caption or default programming can be combined with one of the standard modifiers to retrieve special function key properties. The string value of the programmed key is binary, and might not contain printable characters.

Modifier	Shift	Ctrl	Alt	Modifier	Shift	Ctrl	Alt
0	no	no	no	4000	no	no	yes
1000	yes	no	no	5000	yes	no	yes
2000	no	yes	no	6000	no	yes	yes
3000	yes	yes	no	7000	yes	yes	yes

10000 retrieve the function key caption and tip string

20000 retrieve the default programming for the key (read only)

Virtual Key	Number	Virtual Key	Number
F1	112	Backspace	8
F2	113	Tab	9
F3	114	Insert	45
F4	115	Delete	46
F5	116	Home	36
F6	117	End	35
F7	118	Page Up	33
F8	119	Page Down	34
F9	120	Left	37
F10	121	Right	39
F11	122	Up	38
F12	123	Down	40
F13	124	Escape	27
F14	125	Enter	13
F15	126	Keypad Enter	253
F16	127		

Note: almost all keyboard keys are programmable; see any standard Windows reference for virtual key numbers not shown above.

`FontName` property (string)
This is the name of the terminal font used by the session.

`FontSize` property (integer)
This is the size of the terminal font in points.

`GetBlock` method

scrblk = ***session***.`GetBlock` (***left*** , ***top*** , ***right*** , ***bottom*** , [***page***])

Creates a `ScreenBlock` object containing the contents of the specified block. The returned object contains all text, attribute, color, protect and character set information contained within the block. Default ***page*** is the current page. This method, when used with the `SetBlock` method, may be used to effect "windowing". The return value of this method must be assigned to a variable which has been declared as type `ScreenBlock`.

`GetText` method

text = ***session***`.GetText ([`***col*** `[,` ***row*** `[,` ***cols*** `[,` ***mode*** `]]])`

Returns the text contents of a row on the termianal screen. Default ***col*** and ***row*** is the current cursor location. Default number of columns is to the end of the row. Use zero for ***mode*** to return all normal characters on the row, including protected characters. Use 1 to return non-protected characters only. Add 2 to include line-drawing character and special symbols (Unicode). Non-negative row numbers refer to the active (non-history) area of the terminal screen (zero is the first data row of the active screen). Negative row numbers refer to history rows (row –1 is the last history row).

`GmodeEnable` property (boolean)

Set to True (non-zero) if Tektronix graphics mode is enabled for the session; otherwise set to False (zero).

`Handshake` property (integer)

This is the handshake (flow control) method used by the serial port attached to the session. This property is only meaningful when the communications device type is `atDevSERIAL` or `atDevMODEM`. Acceptable handshake settings are `atHandshakeNONE`, `atHandshakeXON` (inbound only Xon/Xoff), `atHandshakeXIO` (bi-directional Xon/Xoff), `atHandshakeRTS` or `atHandshakeDTR`.

`Hangup` method

result = ***session***`.Hangup()`

Uses the Auto-Dialer to disconnect (hangup) a modem call. The return value specifies the result of the hangup operation. See the `DialStatus` property for a description of the return codes.

`Height` property (long)

This is the session window height in pixels.

`Hide` method

This method makes the session window invisible.

`HistoryRows` property (integer)

This is the number of rows allocated for the session history buffer. If a buffer is allocated, each time a line is scrolled off the screen, the line is placed in the history buffer. Likewise, whenever the screen is cleared, the contents of the screen before clearing is copied to the history buffer.

`HostName` property (string)
This is the name of the host computer system which a session is connected to. This property is only meaningful when the communications device type is telnet, ssh or PicLan. In the case of Telnet and ssh connections, you can use the host IP address in place of the host name. After changing this property, use the `Reset` method for the change to take effect.

`HostPort` property (string)
For telnet and ssh connections, this is the host TCP port number to connect to. If this number is zero, telnet will connect to TCP port 23 and ssh will connect to TCP port 22. For PicLan connections only, this is the Pick pseudo-port number; -1 indicates "any available port".

`HostTermType` property (string)
For telnet and ssh connections, this is the terminal type sent to the host when initiating the connection. This setting is only needed if AccuTerm's default terminal type is not compatible with the host terminal names.

`hWnd` property (long)
This read-only value is the window handle for the session's terminal screen. This handle may be used in Windows API calls in conjunction with the `Declare` statement.

`Icon` property (string)
This is the name of the icon file used for the session.

`ID` property (string)
This is the unique session ID.

`Input` method

result = ***session***.`Input`([***mode*** [, ***maxlen*** [, ***timeout***]]])

This method accepts input from the communications device attached to the session. If ***mode*** is 0 (default), raw data is returned, up to ***maxlen*** characters or until ***timeout*** seconds have passed. If ***mode*** is 1, a "line" of data is returned, up to ***maxlen*** characters or until ***timeout*** seconds have passed. A "line" is terminated by a ***CR***, ***LF***, ***CR+LF*** or ***LF+CR***. The terminator is not returned as part of the input. Default values for ***maxlen*** and ***timeout*** are 80 and 30 respectively.

When using the method, you should set the `InputMode` property to 1 or 2 to ensure all received data is filtered by this function.

This method cannot be used when using late binding; use the `ReadText` method method instead.

`InputMode` property (integer)
This specifies how received data is processed by the terminal session. If the `InputMode` property is set to 0 (normal), any received data is immediately processed by the terminal emulator. Use this mode when the script is not concerned with the data stream. Setting the mode to 1 (synchronous) allows the terminal emulator to process all data which the script has processed. That is, all received data examined by a `WaitFor` method or `Input` method is also processed by the emulator. Finally, setting the mode to 2 (disconnected) disconnects the terminal emulator from the received data stream. It is up to the script to pass any data to the emulator using the `Emulate` method. If a session was created with the `Sessions.Add` method, the initial `InputMode` setting may be specified as an argument.

`KermitCRC` property (integer)
Set to True (non-zero) to enable CRC block checking, else set to False (zero) to use checksum block checking. If this option is enabled, and the host Kermit supports this option, then a 16 bit CRC is used to verify packet integrity.

`KermitEOL` property (integer)
Specifies the ASCII character to be sent at the end of each Kermit packet. Usually 13 (***CR***), but may be 10 (***LF***) for some hosts.

`KermitErrRetry` property (integer)
Specifies the number of times Kermit will retry during a file transfer before the transfer is aborted.

`KermitInitRetry` property (integer)
Specifies the number of times Kermit will retry when initiating a file transfer before the transfer is aborted.

`KermitQuote` property (integer)
Set to True (non-zero) to enable the “eighth bit quoting” option, else set to False (zero). If this option is enabled, and the host Kermit supports this option, then any characters which have the eighth bit set are “quoted” using a sequence of characters which do not have the eighth bit set. This option is useful for connections which do not support eight bit data.

`KermitRept` property (integer)
Set to True (non-zero) to enable the run length encoding data compression option, else set to False (zero). If this option is enabled, and the host Kermit supports this option, then repeated sequences of any character are encoded to compress the data.

`KermitTimeout` property (integer)
Specifies the number of seconds before Kermit file transfer operations time out.

`Left` property (long)
This is the horizontal position of the session window relative to the upper-left corner of the inside of AccuTerm's main window. This value is in pixels.

`LegibleFont` property (boolean)
Set to True (non-zero) to keep fonts legible when scaling fonts to fit screen; otherwise set to False (zero).

`LoadImage` method

***session*.`LoadImage` *filename*, *col*, *row* [, *width* [, *height* [, *preserveaspect* [, *borderstyle*]]]]**

Creates an image on the terminal screen at the location specified by ***col*** and ***row***. ***Filename*** specifies the name or URL of the image file. Image file formats supported are bitmap, Windows metafile, GIF, TIFF (uncompressed), TARGA and JPEG. If ***width*** and ***height*** are specified, then the image is scaled to fit the specified area; otherwise the image is displayed in its actual dimensions. If ***preserveaspect*** is non-zero, then the aspect ratio of the original image is preserved by reducing either width or height. To enclose the image in a border, specify ***borderstyle*** of 0 for none, 1 for simple, 2 for raised or 3 for inset. Multiple images may be loaded on the terminal screen. Images are considered "protected" data; a "clear un-protected characters" command will not clear images from the terminal screen.

`Locate` method

***session*.`Locate` *col*, *row* [, *page*]**

Moves the cursor to the specified screen location and optionally changes pages.

`LockBaudRate` property (boolean)
Set to True (non-zero) to lock the modem to the specified baud rate; otherwise set to False (zero). Note: not all modems support this setting.

`LockFKeys` property (integer)
Set to 0 for unlocked function keys (host can reset or reprogram keys), 1 to lock from reset (host can reprogram keys, but not reset them) or 2 to lock from programming (host cannot reset or reprogram function keys).

`LockKeyboard` property (boolean)
Set to True (non-zero) to prevent keystrokes from being sent to the host system; otherwise set to False (zero).

`MapUpperFKeys` property (boolean)
Setting this property to True (non-zero) causes AccuTerm to map **CTRL+F1** through **CTRL+F10** as **F11** to **F20**. This is the default behavior, since the PC keyboard does not have keys **F13** to **F20**.

`Menu` property (`Menu` object)
This returns a reference to the session `Menu` object, which may be used to customize AccuTerm's menus and toolbars. For more information on the `Menu` object, refer to the chapter titled "Customizing the Menu and Toolbar".

`MessageRows` property (integer)
This property sets the number of message rows and status lines for the session. The value must be between 0 and 3. Note that some emulations do not support any status or message rows, and some only support a single status line. Others support up to 3 lines. For more information, see the **Status Lines** topic in **Terminal Type Settings** in the **Users Guide**.

`MouseEnable` property (boolean)
Set to True (non-zero) to enable the ESC STX "1" AccuTerm mouse-on command; otherwise set to False (zero).

`MouseTableAdd` method
***session*.`MouseTableAdd` *button* [, *pattern* [, *click* [, *dblclk*]]]**
Adds an entry to the mouse pattern table for the session. The mouse pattern table is used to associate patterns on the screen with responses to be sent to the host when the mouse is clicked. ***Button*** is 1 for left button, 2 for right button and 3 for middle button. ***Pattern*** is a string containing a "regular

expression" describing the pattern to be matched. ***Click*** is a string, and is sent to the host when the specified button is clicked over the pattern. ***Dblclk*** is the response if the button is double-clicked over the pattern. See chapter titled "Customizing the Menu and Toolbar" for details on using the mouse pattern table feature.

`MouseTableLoad` method

session`.MouseTableLoad` ***filename***

Loads a mouse pattern table from a file. See chapter titled "Customizing the Menu and Toolbar" for the format of the mouse pattern table file.

`MouseTableReset` method

Clears the mouse pattern table.

`Move` method

session`.Move` ***left*** [`,` ***top*** [`,` ***width*** [`,` ***height***]]]

Repositions and resizes the session window. The positions and dimensions are specified in pixels, and are relative to the upper-left corner of the inside of AccuTerm's main window.

`NoAutoWrap` property (boolean)

Setting this option disables the automatic wrapping at the end of a line.

`NormCols`, `NormRows` properties (integer)

These properties define the terminal screen dimensions when the terminal is placed into "normal mode", also known as "80 column mode".

`NormMode` method

Causes the session to switch to normal (80 column) mode.

`Output` method

session`.Output` ***expr***

This method transmits data to the host using the communications device attached to the session. Transmits ***expr*** (a string expression) to the connected host computer. This method is the same as the `WriteText` method.

`OverrideModemConfig` property (boolean)

Set to True (non-zero) to override modem Control Panel settings for baud rate, data bits, stop bits, parity and handshake with current session settings for these properties; otherwise set to False (zero).

`Page` property (integer)
This specifies the number of the current video page for the session. The value must be between 0 and the number of pages allocated for the session minus 1 (see `Pages` property).

`Pages` property (integer)
This specifies the number of video pages for the session. The value must be between 1 and 25.

`Palette()` property (array of OLE_COLOR)
This array contains the color values for the 16 palette entries for the session. The default color assignment for each palette entry is shown in the table for the `Color` property as "foreground color." Use the `RGB()` function to produce an OLE_COLOR.

An additional 6 palette entries exist for drawing borders (see the **Visual Styles** topic in **Color Settings** in the **User Guide** for more information). These are:

16 = flat outer border
17 = flat inner border
18 = shadow outer border
19 = shadow inner border
20 = highlight outer border
21 = highlight inner border

`Parity` property (integer)
This is the parity setting used by the serial port attached to the session. This property is only meaningful when the communication device type is `atDevSERIAL` or `atDevMODEM`. Acceptable values are atParityNONE, atParityEVEN, atParityODD, atParityMARK and atParitySPACE. After changing this property, use the `Reset` method for the change to take effect.

`Paste` method
***session*`.Paste [` *filename*]
Copies the text contents of the clipboard (or file if ***filename*** is specified) to the host.

`PasteEOFChar` property (integer)
This is the ASCII character which is transmitted to the host at the end of paste operaton. If this value is zero, no character will be transmitted.

`PasteEOFMode` property (integer)
Set to 0 if no character is transmitted at the end of a paste operation. Set to 1 if a ***SUB*** control code is transmitted. Set to 2 to use the character defined by the `PasteEOFChar` property.

`PasteEOLChar` property (integer)
This is the ASCII character which is transmitted to the host at the end of each line in a paste operaton. If this value is zero, no character will be transmitted.

`PasteEOLMode` property (integer)
Set to 0 if a ***CR*** is transmitted at the end of each line of a paste operation. Set to 1 if a ***LF*** is transmitted. Set to 2 if a ***CR+LF*** is transmitted. Set to 3 if a ***TAB*** is transmitted. Set to 4 if no character is transmitted. Set to 5 to use the character defined by the `PasteEOLChar` property. If you want to suppress the end-of-line character after the *last* line, add 128 to this value.

`PlayMidi` method
session**.** `PlayMidi` ***filename***
Plays the MIDI sound file specified by ***filename***. ***Filename*** may specify a local file or a URL.

`PlayWave` method
session**.** `PlayWave` ***filename***
Plays the Wave sound file specified by ***filename***. ***Filename*** may specify a local file or a URL.

`Port` property (integer)
Specifies the Com port number attached to the session. This property is only meaningful when the communication device type is `atDevSERIAL`. Acceptable values are from 1 to 127. After changing this property, use the `Reset` method for the change to take effect.

`PrinterClose` method
session.`PrinterClose`
Closes the current print job.

`PrinterColorMode` property (integer)
If this property is non-zero and the ScreenPrintMode property is non-zero (Graphics mode), the screen is printed in color; otherwise the screen is printed in black and white. This property has no effect when the ScreenPrintMode property is zero (Text mode).

`PrinterFontBold` property (boolean)
Set this property to True (non-zero) to print using boldface. This property affects slave (or Aux) print jobs when the SlavePrintMode property is 1 (Graphics mode). This property has no effect when the SlavePrintMode property is zero (Text mode).

`PrinterFontItalic` property (boolean)
Set this property to True (non-zero) to print using italics. This property affects slave (or Aux) print jobs when the SlavePrintMode property is 1 (Graphics mode). This property has no effect when the SlavePrintMode property is zero (Text mode).

`PrinterFontName` property (string)
Set this property to the name of the font to use for slave (or Aux) print jobs when the SlavePrintMode property is 1 (Graphics mode). This property has no effect when the SlavePrintMode property is zero (Text mode).

`PrinterFontSize` property (integer)
Set this property to the size (in points) of the font to use for slave (or Aux) print jobs when the SlavePrintMode property is 1 (Graphics mode). This property has no effect when the SlavePrintMode property is zero (Text mode).

`PrinterMode` property (integer)
This is the state of the auxiliary printer: 0 if printer is off, 1 if printer is auto-print mode, 2 if the printer is in transparent print mode.

`PrinterName` property (string)
This is the name of the printer used by the session for slave printer (aux port) output. It may be set to null ("") to indicate the default printer, or "no printer" to disable slave printing. Otherwise specify the name of the printer as defined in Windows printer settings. *Note: network printers require UNC names.*

`PrinterOff` method
session`.PrinterOff`
Turns off the auxiliary printer.

`PrinterOn` method

session`.PrinterOn` [***mode***]

Turns on the auxiliary printer. If ***mode*** is zero (default), printer is in "copy" or "auto print" mode (data is displayed on both the screen and printer). Otherwise, printer is in "transparent" mode (data is sent to the printer only, not the screen).

`PrinterOrientation` property (integer)

Set this property to 0 use the default page orientation, 1 for portrait or 2 for landscape. This property applies only to slave (or Aux) print jobs and only when the SlavePrintMode property is 1 (Graphics mode). This property has no effect when the SlavePrintMode property is zero (Text mode).

`PrinterPaperSize` property (integer)

Set this property to 0 use the default paper size for the printer. Otherwise use a paper size from the table below. This property applies only to slave (or Aux) print jobs and only when the SlavePrintMode property is 1 (Graphics mode). This property has no effect when the SlavePrintMode property is zero (Text mode).

1 Letter 8 1/2 x 11 in
2 Letter Small 8 1/2 x 11 in
3 Tabloid 11 x 17 in
4 Ledger 17 x 11 in
5 Legal 8 1/2 x 14 in
6 Statement 5 1/2 x 8 1/2 in
7 Executive 7 1/4 x 10 1/2 in
8 A3 297 x 420 mm
9 A4 210 x 297 mm
10 A4 Small 210 x 297 mm
11 A5 148 x 210 mm
12 B4 (JIS) 250 x 354
13 B5 (JIS) 182 x 257 mm
14 Folio 8 1/2 x 13 in
15 Quarto 215 x 275 mm
16 10x14 in
17 11x17 in
18 Note 8 1/2 x 11 in
19 Envelope #9 3 7/8 x 8 7/8
20 Envelope #10 4 1/8 x 9 1/2
21 Envelope #11 4 1/2 x 10 3/8
22 Envelope #12 4 \276 x 11
23 Envelope #14 5 x 11 ½
24 C size sheet
25 D size sheet
26 E size sheet
27 Envelope DL 110 x 220mm
28 Envelope C5 162 x 229 mm
29 Envelope C3 324 x 458 mm
30 Envelope C4 229 x 324 mm
31 Envelope C6 114 x 162 mm
32 Envelope C65 114 x 229 mm
33 Envelope B4 250 x 353 mm
34 Envelope B5 176 x 250 mm
35 Envelope B6 176 x 125 mm
36 Envelope 110 x 230 mm
37 Envelope Monarch 3.875x7.5 in
38 6 3/4 Envelope 3 5/8 x 6 1/2 in
39 US Std Fanfold 14 7/8 x 11 in
40 German Std Fanfold 8 1/2 x 12 in
41 German Legal Fanfold 8 1/2x13 in

`PrinterPaperSource` property (integer)
Set this property to 0 use the default paper source for the printer. Otherwise use a paper source from the table below. This property applies only to slave (or Aux) print jobs and only when the SlavePrintMode property is 1 (Graphics mode). This property has no effect when the SlavePrintMode property is zero (Text mode).

1 Upper tray
2 Lower tray
3 Middle tray
4 Manual feed
5 Envelope feeder
6 Envelope manual
7 Auto
8 Tractor

`PrinterTimeout` property (integer)
This property specifies the number of seconds of inactivity before a print job is closed.

`PrintJobEject` property (integer)
This property specifies if a page is ejected before or after a print job. Set this property to 0 for no page ejects, 1 to eject page before print job, 2 to eject page after print job or 3 to eject page before and after print job.

`PrintScreen` method
Prints the text screen to the printer.

`PrintScreenBackground` property (boolean)
Set to True (non-zero) to print the screen background when using the Print Screen command and the PrintScreenMode property is non-zero (Graphics mode); otherwise set to False (zero). This property has no effect when the ScreenPrintMode property is zero (Text mode).

`PrintScreenEject` property (integer)
This property specifies if a page is ejected before or after a print screen. Set this property to 0 for no page ejects, 1 to eject page before print screen, 2 to eject page after print screen or 3 to eject page before and after print screen.

`ProtectAttr` property (integer)
Visual attribute number assigned to “protected fields” under Wyse/ADDS emulation. See `Colors()` property for a list of attribute numbers.

`ReadText` method

result = ***session***.`ReadText`([***mode*** [, ***maxlen*** [, ***timeout***]]])

This method accepts input from the communications device attached to the session. If ***mode*** is 0 (default), raw data is returned, up to ***maxlen*** characters or until ***timeout*** seconds have passed. If ***mode*** is 1, a "line" of data is returned, up to ***maxlen*** characters or until ***timeout*** seconds have passed. A "line" is terminated by a ***CR***, ***LF***, ***CR+LF*** or ***LF+CR***. The terminator is not returned as part of the input. Default values for ***maxlen*** and ***timeout*** are 80 and 30 respectively. This method is the same as the `Input` method, and is available when using late binding.

When using the method, you should set the `InputMode` property to 1 or 2 to ensure all received data is filtered by this function.

`Reset` method

session.`Reset` [***mode***]

Resets the terminal (***mode*** = 1), the communication device (***mode*** = 2) or both (default, ***mode*** = 0). Note: you may need to reset the communication device after changing certain device related properties in order for the changes to take effect.

`ResetComm` method

session.`ResetComm`

Resets the communication device. *Note: you may need to reset the communication device after changing certain device related properties in order for the changes to take effect.*

`ResetTerm` method

Resets the terminal emulator for session (clears screen, resets protect mode, unlocks keyboard, etc.)

`Resize` method

session.`Resize` ***width***, ***height***

Resizes the session window. The dimensions are specified in pixels.

`Save` method

session.`Save` [***filename***]

Saves the session configuration. If ***filename*** is not specified, the original file name is used. If there is no file name for the session, the user will be prompted for a file name.

`SaveAs` method
Saves the session configuration prompting for the file name.

`ScaleFont` property (boolean)
Set to True (non-zero) to enable automatic font scaling; otherwise set to False (zero).

`ScreenPrintMode` property (boolean)
Set to False (zero) for text mode, True (non-zero) for graphics mode. When this property is True, the PrinterColorMode and PrintScreenBackground properties affect how the screen is printed.

`ScrMode` property (integer)
Set to zero for normal (80 column) mode, 1 for extended (132 column) mode.

`ScrollHistory` method

session*.`ScrollHistory` *cmd

This method manipulates the history scroll position. ***Cmd*** = `atScrlHistLineUp` (0) to scroll the history up one line; `atScrlHistLineDown` (1) to scroll the history down one line, `atScrlHistPageUp` (2) to scroll up one page, `atScrlHistPageDown` (3) to scroll down one page, `atScrlHistTop` (6) to scroll to the top (first line) of the history, `atScrlHistBottom` (7) to scroll to the bottom (live screen).

`ScrollMode` property (integer)
This controls the appearance of the vertical scroll bar: 0=no scroll bar (scroll bar is only visible if the current number of Rows will not fit in the current window size), 1=always show scroll bar, and 2=automatically show scroll bar when cursor is positioned near right border of window.

`Select` method

***session*.`Select` [*left* [, *top* [, *right* [, *bottom*]]]]**

Places a selection rectangle on the session screen. Defaults are upper left corner and lower right corner of screen.

This method is the same as the `SetSelection` method, but is *not* available when using late binding.

`Selection` property (string)
Returns the current selection, if any. Lines are separated by ***CR LF***. If there is no current selection, returns null ("").

`SetBlock` method

session `.SetBlock` ***ScreenBlock*** `,` ***left*** `,` ***top*** `,` [***page***]
Copies the contents of `ScreenBlock` object ***ScreenBlock*** to the specified location on the screen. The copied block contains all text, attribute, color, protect and character set information. Default ***page*** is the current page. This method must be used with the `GetBlock` method, and may be used to effect "windowing".

`SetExtension` method

session `.SetExtension` ***leadin*** `,` ***terminator***
If ***leadin*** is not NULL, then this method enables the session `Extension` event. This event may be used to extend the functionality of AccuTerm's emulation engine. The ***leadin*** string specifies one or more "lead-in" character codes, which, when preceded by the ***ESC*** control code, signal the start of an "extended" function. Receipt of one of the characters in the ***terminator*** string signals the end of the "extended" function. If ***terminator*** is NULL, then the next character after one of the ***leadin*** characters signals the end of the "extended" function. When the `Extension` event is fired, the string beginning with the ***leadin*** character and ending with the ***terminator*** character is passed to the event handler. If leadin is NULL, then the `Extension` event is disabled. *Note: only characters which form invalid terminal escape sequences should be used for* ***leadin****, otherwise the extension will not function.*

`SetSelection` method

session `.SetSelection` [***left*** [`,` ***top*** [`,` ***right*** [`,` ***bottom***]]]]
Places a selection rectangle on the session screen. Defaults are upper left corner and lower right corner of screen.

This method is the same as the `Select` method, but is available when using late binding.

`SetText` method

session `.SetText` ***text*** [`,` ***col*** [`,` ***row*** [`,` ***color***]]])
Copies ***text*** (a string expression) to the specified location on the session screen. Default location is the current cursor location. Default color is last

color used, or value assigned to the `Color` property. The cursor location is updated by this method.

`Settings` method (`Settings` object)
This object is used to set session settings as a block. It is intended to be used in a `With Settings . . . End With` structure. When used in this manner, all of the settings which are modified are applied to the session when the `End With` statement is executed. This prevents errors when certain settings depend on others (such as `Port` and `Baud`) and also improves performance. Most of the `Session` properties may be set using this object.

Note: when this method is used in a .NET language, you should call the Apply method before the final End With because of non-deterministic object termination in that environment.

`Show` method
Makes the session window visible.

`ShowErrs` property (boolean)
Set to True (non-zero) to enable notification of communications errors; otherwise set to False (zero).

`SlavePrintMode` property (integer)
Set to 0 for text mode, 1 for graphics mode. When this property is 1, the PrinterFont…, PrinterOrientation, PrinterPaperSize and PrinterPaperSource properties affect how the slave (Aux) print job is printed. This property can also be set to the ID of any installed "print adapter" script. Print adapter scripts are used to process the print job using an external program, such as PrintWizard. See the Users Guide, Printer Settings, for more information on print adapters. You can easily create your own print adapter scripts for custom print job processing – just use one of the included scripts, such as PrintWizardPCL.atsc, as an example.

`Sound` property (string)
This property may be used to specify a custom sound for the terminal beep. Set this property to null to use the default sound. Set to a valid .wav file name to play the specified wave file. Set to “SystemDefault”, “SystemHand”, “SystemExclamation” or “SystemAsterisk” to use the sound associated with a system event (as defined in the Control Panel Sounds applet). Set to *frequency,duration* to use a true “beep” at the specified frequency (Hz) and duration (ms).

`SSHAuth` property (integer)
This property may be used to select the authentication method which will be used by a Secrue Shell (ssh) session to authenticate a user to the host. Available authentication methods are public key (2) and password (3). Use zero (0) for the default authentication method (password).

`SSHBreakCharacter` property (integer)
This property specifies the ASCII code of a character which is used to indicate a Break signal to the host. The default value is 3 (**CTRL+C**).

`SSHCipher` property (integer)
This property may be used to select a cipher to be used to encrypt a Secure Shell (ssh) session. Available ciphers are: triple DES (3), blowfish (6), 128 bit AES (7) and 256 bit AES (8). Use zero (0) for the default cipher (triple DES). *AES encryption is only available for SSH2 and requires Windows XP.*

`SSHKeepalive` property (boolean)
If this setting is True (non-zero), AccuTerm will send a special "keepalive" message periodically to maintain the connection (some hosts and routers automatically disconnect when they detect an idle connection).

`SSHKey` property (string)
This property contains the name of the SSH private key file. If no path is included, AccuTerm's default key folder will be used to read the key file.

`SSHNoDelay` property (boolean)
Normally, AccuTerm delays outbound network messages for a short time in order to "coalesce" multiple small messages into a single packet. When this setting is True (non-zero), AccuTerm will send each message as soon as possible. Setting this option may improve throughput, especially when running in AccuTerm's GUI environment and during file transfers, at the expense of increased network traffic.

`SSHVersion` property (integer)
This property may be used to select the desired SSH protocol version. Set this property to zero (0) for automatic protocol selection. Set to 1 for SSH version 1 or set to 2 for SSH version 2. When automatic protocol selection is used, AccuTerm selects the highest protocol version supported by the host.

`StopBits` property (integer)
This is the number of stop bits used by the serial port attached to the session. This property is only meaningful when the communication device type is `atDevSERIAL` or `atDevMODEM`. Acceptable values are 1 or 2. After changing this property, use the `Reset` method for the change to take effect.

`Strip8th` property (boolean)
Set to True (non-zero) to cause AccuTerm to truncate received data to 7 bits; otherwise set to False (zero).

`TelnetAltBreak` property (boolean)
Set to True (non-zero) to send the Telnet Break command when the **BREAK** key is pressed; otherwise set to False (zero) for to send the Telnet Interrupt Process command.

`TelnetBinary` property (boolean)
Set to True (non-zero) to enable Telnet binary communication mode; otherwise set to False (zero) for text communication mode.

`TelnetBypass` property (boolean)
Set to True (non-zero) to bypass initial option negotiation; otherwise set to False (zero) for normal negotiation. This is required for certain hosts like Pick's D3 which do not implement the complete Telnet protocol.

`TelnetEcho` property (boolean)
Set to True (non-zero) to enable Telnet remote-echo mode; otherwise set to False (zero) for local echo mode. Note: when using local echo, you should also set the `Duplex` property to "half".

`TelnetKeepalive` property (boolean)
If this setting is True (non-zero), AccuTerm will send a special "keepalive" message periodically to maintain the connection (some hosts and routers automatically disconnect when they detect an idle connection). *Note: this setting should not be used when connecting to a D3/NT host.*

`TelnetNoDelay` property (boolean)
Normally, AccuTerm delays outbound network messages for a short time in order to "coalesce" multiple small messages into a single packet. When this setting is True (non-zero), AccuTerm will send each message as soon as possible. Setting this option may improve throughput, especially when running in AccuTerm's GUI environment and during file transfers, at the expense of increased network traffic.

`Terminate` method

session.`Terminate` [***prompt***]

This closes the session. If ***prompt*** = 1 or 3, and if any session settings have been modified subsequent to loading or saving the session, the user will be prompted whether to save the settings. If ***prompt*** = 2 or 3, and the session is connected via a network or dialup connection, the user will be prompted to disconnect.

This method is the same as the `Close` method, and is available when using late binding.

`TermType` property (integer)
This is the terminal emulation setting for the session. Possible values are `atTermTTY` (basic TTY), `atTermVPA2` (ADDS Viewpoint A2), `atTermVP60` (ADDS Viewpoint 60), `atTermP60` (ProComm Viewpoint 60), `atTermA2E` (ADDS Viewpoint A2 Enhanced), `atTermWY50` (Wyse 50), `atTermWY60` (Wyse 60), `atTermVT52` (DEC VT52), `atTermVT100` (DEC VT100), `atTermVT220` (DEC VT220), `atTermVT320` (DEC VT320), `atTermVT420` (DEC VT420), `atTermLinux` (Linux Console), `atTermSCO` (SCO Unix Console), `atTermANSI` (ANSI BBS), `atTermPICKMON` (Pick PC Monitor), `atTermP30` (MDIS P30) or `atTermTEK` (Tektronix 4014).

`Top` property (long)
This is the vertical position of the session window relative to the upper-left inside corner of AccuTerm's main window. This value is in pixels.

`U2DeviceLicensing` property (boolean)
If this setting is True (non-zero), multiple connections (sessions) to Enterprise versions of UniData and UniVerse consume only a single user license. Contact your IBM dealer for information regarding device licensing.

`UnloadImage` method

***session*.`UnloadImage` [*filename*]**

Removes an image created with the `LoadImage` method from the terminal screen. The image to be removed is identified by ***filename***. If ***filename*** is not specified, the all images are removed.

`Upload` method

***result* = *session*.`Upload`(*source* , *protocol* , *binary* [, *overwrite*])**

Uploads a file from the user's PC to the connected host system. ***Source*** is the name of the source file(s) (Kermit, Ymodem and Zmodem may use wild-card characters in ***source***). ***Protocol*** is `atProtocolASCII`, `atProtocolKermit`, `atProtocolXmodem`, `atProtocolYmodem` or `atProtocolZmodem`. ***Binary*** is True (non-zero) for binary transfer mode, False (zero) for text transfer mode. ***Overwrite*** is meaningful for Zmodem only and must be `atProtect`, `atOverwrite`, `atAppend`, `atNewer`, `atUpdate` or `atResume`.

`Visible` property (boolean)

Set to True (non-zero) to make the session window visible. Set to False to hide the session window.

`WaitFor` method

***result* = *session*.`WaitFor`(*mode*, *timeout*, *string1* [, ... , *string10*])**

This method causes AccuTerm to wait for one or more strings to be received from the host system. Returns the index of the string first matched, or zero (0) if no string matched within ***timeout*** seconds. If ***mode*** is zero, a case-sensitive comparison is performed, otherwise the comparison is case-insensitive. Up to ten target strings may be specified.

When using the method, you should set the `InputMode` property to 1 or 2 to ensure all received data is filtered by this function.

`Width` property (long)

This is the session window width in pixels.

`WindowState` property (integer)

This is the state of the session window. Set to 0 for normal, 1 for minimized or 2 for maximized.

`WriteText` method

session*.`WriteText` *expr

This method transmits data to the host using the communications device attached to the session. Transmits ***expr*** (a string expression) to the connected host computer.

This method is the same as the `Output` method.

`XferBytes` property (long integer)

This read-only property is the number of bytes transferred as a result of the last file transfer operation.

`XferFiles` property (long integer)

This read-only property is the number of files transferred as a result of the last file transfer operation.

`XferStatus` property (integer)

This read-only property reflects the status of the last `Upload` or `Download` method. Status codes are described in the following table:

Status	Description
-1	File transfer in progress.
0	File transfer successful.
1	Invalid source file or invalid destination directory.
2	File transfer aborted by user.
3	Destination file already exists.
4	File transfer timed out.
7	File transfer protocol failure.
8	X/Y/ZModem require 8 data bits.
9	X/YModem require hardware flow control.
10	Destination file is write protected

`XmodemTimeout` property (integer)

Specifies the number of seconds before Xmodem file transfer operations time out.

`YmodemTimeout` property (integer)

Specifies the number of seconds before Ymodem file transfer operations time out.

`ZmodemAuto` property (integer)
Set to True (non-zero) to enable automatic Zmodem downloads, otherwise set to False (zero).

`ZmodemTimeout` property (integer)
Specifies the number of seconds before Zmodem file transfer operations time out.

The Settings Object
The `Settings` property of the `Session` object returns a reference to a temporary `Settings` object. This temporary object is used to retrieve or modify a number of settings as a group. When changing multiple properties, using the `Settings` object in a `With . . . End With` construct improves efficiency.

Nearly all of the properties of the `Settings` object are identical to properties of the `Session` object or to properties of the `AccuTerm` object.

`Apply` method
The `Apply` method applies any changed properties to the emulator.

`Ansi8Bit` property (boolean)
If this setting is non-zero, AccuTerm sends 8 bit control codes. Othewise the eqivalent 7-bit escape sequence is sent. This property is only effective when AccuTerm is emulating one of the VT terminals.

`AnsiAppCursor` property (boolean)
If this setting is non-zero, AccuTerm sends "application codes" instead of "cursor codes" when cursor keys are pressed. This property is only effective when AccuTerm is emulating one of the VT terminals.

`AnsiAppKeypad` property (boolean)
If this setting is non-zero, AccuTerm sends "application codes" instead of numeric characters when key on the numeric keypad are pressed. This property is only effective when AccuTerm is emulating one of the VT terminals.

`AnsiAutoPrint` property (boolean)
If this setting is non-zero, "copy" or "auto print" printing to the slave printer works just like a real VT terminal. When set to zero, "copy" or "auto print" works like a Wyse terminal.

`Answerback` property (string)
This is the string that AccuTerm returns to the host system when the host requests the "answerback message".

`AsciiDelay` property (integer)
Specifies the interline delay used for ASCII file uploads in milliseconds.

`AsciiEOL` property (integer)
Specifies the line ending sequence for ASCII file uploads. Set to 0 for ***CR***, 1 for ***LF*** or 2 for ***CR+LF***.

`AttributeMask` property (integer)
The `AttributeMask` property indicates which visual effects are to be displayed. Use 2 for blink , 8 for underline, 10 for both blink and underline, or zero to disable both blinking and underline. The `AttributeMask` property also controls whether border effects are available, as well as the border size. To enable border effects, add 64 for thin internal borders, 128 for thick internal borders, 576 for thin external borders and 640 for thick external borders. When internal borders are selected, the vertical border lines are drawn inside the character cell; when external borders are selected, the vertical border lines are drawn in adjacent cells.

`AutoAnswer` property (boolean)
If this setting is non-zero, AccuTerm will answer incoming calls when the session is connected to a modem.

`Baud` property (integer)
This is the baud rate used by the serial port attached to the session. This property is only meaningful when the communication device type is "Serial Port". Acceptable baud rates are `atBaud300`, `atBaud1200`, `atBaud2400`, `atBaud9600`, `atBaud14400`, `atBaud19200`, `atBaud38400`, `atBaud57600` and `atBaud115200`.

`BkspSendsDel` property (boolean)
If this setting is True (non-zero), pressing the Backspace key causes AccuTerm to send the ***DEL*** control code. Otherwise, AccuTerm sends the ***BS*** control code.

`BoldFont` property (boolean)
Set to True (non-zero) if terminal font is bold, False (zero) if font is normal.

`Changed` property (boolean)
If this value is True (non-zero), then the settings have been changed.

`CharacterMapping` property (string)
Specifies custom character set mapping. Currently only the Euro currency symbol can be specified; set this property to "EURO SIGN=nnn" where nnn is the ASCII character used to represent the Euro currency symbol.

`Charset` property (string)
The character set used by the host. The only valid values are NULL for the default character set for the selected terminal emulation, or "NATIVE" if the host uses the current 8-bit (non-Unicode) Windows character set.

`Colors()` property (array of integer)
This array maps foreground and background colors (using the color index described above) to visual attribute combinations. See `Session Colors` property for more details.

`ConnectTimeout` property (integer)
This is the number of seconds to wait while attempting to connect before a timeout error occurs.

`CursorType` property (boolean)
Set to True (non-zero) if cursor shown as a block, False (zero) if cursor shown as an underline.

`DataBits` property (integer)
This is the number of data bits used by the serial port attached to the session. This property is only meaningful when the communication device type is `atDevSERIAL` or `atDevMODEM`. Acceptable values are 7 or 8. After changing this property, use the `Reset` method for the change to take effect.

`DefaultCaptureDir` property (string)
This is the default destination directory used for file capture operations which do not specify a directory.

`DefaultXferDir` property (string)
This is the default destination directory used for file transfer operations which do not specify a directory.

`DefaultXferMode` property (integer)
This is default transfer mode: 0 for text, 1 for binary.

`DefaultXferOverwrite` property (integer)
This is default overwrite setting for received files. Set to True (non-zero) to allow overwrites, else set to False (zero).

`Device` property (integer)
This is the communications device type attached to the session. The device type may be `atDevNONE` for no device (disconnects session), `atDevSERIAL`, `atDevPICLAN`, `atDevTELNET`, `atDevSSH` or `atDevMODEM`.

`Dialog` method

***settings*`.Dialog [`*Title*`, [`*InitTab*`, [`*Tabs*` ]]]`**

This method displays a tabbed settings dialog box which will allow the user to modify most settings. You can specify an optional ***Title*** for the dialog, as well as the initially selected tab number ***InitTab*** (first tab is 0), and a list of ***Tabs*** which are displayed. The ***Tabs*** list is a string with a single letter for each visible tab. Letters which may be used for ***Tabs*** are:

P = printer
X = file transfer
D = device (connection)
T = term type
S = screen
F = fonts
C = colors
K = keyboard
M = miscellaneous

`DisableAppMode` property (boolean)
Set to True (non-zero) to prevent AccuTerm from entering "keypad application mode" or "cursor application mode" when running one of the VT emulations. Set to False (zero) to allow application mode.

`Duplex` property (integer)
This property sets the communications duplex mode to local (no communication to or from host: `atDuplexLOCAL`), full (remote echo: `atDuplexFULL`) or half (local echo: `atDuplexHALF`).

`ExtCols`, `ExtRows` properties (integer)
These properties define the terminal screen dimensions when the terminal is placed into "extended mode", also known as "132 column mode".

`FKeys` property (collection)

`Set` ***settings*`.Fkeys =`** ***collection***
`Set` ***collection*** **=** ***settings*`.Fkeys()`**

This property gets or sets all of the programmed function keys. The collection contains one item for each programmed key. The collection key is the function key index (see `Session FKeys` property). Each item is a string consisting of the key index followed by a NUL character followed by the key contents (string of characters). The key contents may contain control characters.

`FontName` property (string)
This is the name of the terminal font used by the session.

`FontSize` property (integer)
This is the size of the terminal font in points.

`GmodeEnable` property (boolean)
Set to True (non-zero) if Tektronix graphics mode is enabled for the session; otherwise set to False (zero).

`Handshake` property (integer)
This is the handshake (flow control) method used by the serial port attached to the session. This property is only meaningful when the communications device type is `atDevSERIAL` or `atDevModem`. Acceptable handshake settings are `atHandshakeNONE`, `atHandshakeXON` (inbound only Xon/Xoff), `atHandshakeXIO` (bi-directional Xon/Xoff), `atHandshakeRTS` or `atHandshakeDTR`.

`HistoryRows` property (integer)
This is the number of rows allocated for the session history buffer. If a buffer is allocated, each time a line is scrolled off the screen, the line is placed in the history buffer. Likewise, whenever the screen is cleared, the contents of the screen before clearing is copied to the history buffer.

`HostName` property (string)
This is the name of the host computer system which a session is connected to. This property is only meaningful when the communications device type is "PicLan" or "Telnet". In the case of Telnet connections, you can use the host IP address in place of the host name. After changing this property, use the `Reset` method for the change to take effect.

`HostPort` property (string)
For Telnet connections, this is the host TCP/IP port number assigned for Telnet services. For PicLan connections, it is the Pick pseudo-port number; when -1 (negative one) is used, it means "any available port".

`HostTermType` property (string)
For Telnet connections, this is the terminal type sent to the host when initiating the connection. This setting is only needed if AccuTerm's default terminal type is not compatible with the host terminal names.

`Initialize` method
This method returns all emulator settings to their default values.

`KermitCRC` property (boolean)
Set to True (non-zero) to enable CRC block checking, else set to False (zero) to use checksum block checking. If this option is enabled, and the host Kermit supports this option, then a 16 bit CRC is used to verify packet integrity.

`KermitEOL` property (integer)
Specifies the ASCII character to be sent at the end of each Kermit packet. Usually 13 (***CR***), but may be 10 (***LF***) for some hosts.

`KermitErrRetry` property (integer)
Specifies the number of times Kermit will retry during a file transfer before the transfer is aborted.

`KermitInitRetry` property (integer)
Specifies the number of times Kermit will retry when initiating a file transfer before the transfer is aborted.

`KermitQuote` property (boolen)
Set to True (non-zero) to enable the "eighth bit quoting" option, else set to False (zero). If this option is enabled, and the host Kermit supports this option, then any characters which have the eighth bit set are "quoted" using a sequence of characters which do not have the eighth bit set. This option is useful for connections which do not support eight bit data.

`KermitRept` property (boolean)
Set to True (non-zero) to enable the run length encoding data compression option, else set to False (zero). If this option is enabled, and the host Kermit supports this option, then repeated sequences of any character are encoded to compress the data.

`KermitTimeout` property (integer)
Specifies the number of seconds before Kermit file transfer operations time out.

`LegibleFont` property (boolean)
Set to True (non-zero) to keep fonts legible when scaling fonts to fit screen; otherwise set to False (zero).

`Load` method

settings*.`Load` *filename

This method loads all of the emulator settings from the specified INI file.

`LockBaudRate` property (boolean)
Set to True (non-zero) to lock the modem to the specified baud rate; otherwise set to False (zero). Note: not all modems support this setting.

`LockFKeys` property (integer)
Set to 0 for unlocked function keys (host can reset or reprogram keys), 1 to lock from reset (host can reprogram keys, but not reset them) or 2 to lock from programming (host cannot reset or reprogram function keys).

`MapUpperFKeys` property (boolean)
Setting this property to True (non-zero) causes AccuTerm to map **CTRL+F1** through **CTRL+F10** as **F11** to **F20**. This is the default behavior, since the PC keyboard does not have keys **F13** to **F20**.

`MessageRows` property (integer)
This property sets the number of message rows and status lines for the session. The value must be between 0 and 3. Note that some emulations do not support any status or message rows, and some only support a single status line. Others support up to 3 lines. For more information, see the **Status Lines** topic in **Terminal Type Settings** in the **Users Guide**.

`MouseEnable` property (boolean)
Set to True (non-zero) to enable the ESC STX "1" AccuTerm mouse-on command; otherwise set to False (zero).

`NoAutoWrap` property (boolean)
Setting this option disables the automatic wrapping at the end of a line.

`NormCols`, `NormRows` properties (integer)
These properties define the terminal screen dimensions when the terminal is placed into “normal mode”, also known as “80 column mode”.

`OverrideModemConfig` property (boolean)
Set to True (non-zero) to override modem Control Panel settings for baud rate, data bits, stop bits, parity and handshake with current session settings for these properties; otherwise set to False (zero).

`Pages` property (integer)
This specifies the number of video pages for the session. The value must be between 1 and 25.

`Palette()` property (array of OLE_COLOR)
This array contains the color values for the 16 palette entries for the session. The default color assignment for each palette entry is shown in the table for the `Color` property as "foreground color." Use the `RGB()` function to produce an OLE_COLOR.

`Parity` property (integer)
This is the parity setting used by the serial port attached to the session. This property is only meaningful when the communication device type is `atDevSERIAL` or `atDevMODEM`. Acceptable values are atParityNONE, atParityEVEN, atParityODD, atParityMARK and atParitySPACE. After changing this property, use the `Reset` method for the change to take effect.

`PasteEOFChar` property (integer)
This is the ASCII character which is transmitted to the host at the end of paste operaton. If this value is zero, no character will be transmitted.

`PasteEOFMode` property (integer)
Set to 0 if no character is transmitted at the end of a paste operation. Set to 1 if a ***SUB*** control code is transmitted. Set to 2 to use the character defined by the `PasteEOFChar` property.

`PasteEOLChar` property (integer)
This is the ASCII character which is transmitted to the host at the end of each line in a paste operaton. If this value is zero, no character will be transmitted.

`PasteEOLMode` property (integer)
Set to 0 if a ***CR*** is transmitted at the end of each line of a paste operation. Set to 1 if a ***LF*** is transmitted. Set to 2 if a ***CR+LF*** is transmitted. Set to 3 if a ***TAB*** is transmitted. Set to 4 if no character is transmitted. Set to 5 to use the character defined by the `PasteEOLChar` property.

`Port` property (integer)
Specifies the Com port number attached to the session. This property is only meaningful when the communication device type is `atDevSERIAL`. Acceptable values are from 1 to 128. After changing this property, use the `Reset` method for the change to take effect.

`PrinterColorMode` property (integer)
If this property is non-zero and the `ScreenPrintMode` property is non-zero (Graphics mode), the screen is printed in color; otherwise the screen is printed in black and white. This property has no effect when the `ScreenPrintMode` property is zero (Text mode).

`PrinterFontBold` property (boolean)
Set this property to True (non-zero) to print using boldface. This property affects slave (or Aux) print jobs when the `SlavePrintMode` property is 1 (Graphics mode). This property has no effect when the `SlavePrintMode` property is zero (Text mode).

`PrinterFontItalic` property (boolean)
Set this property to True (non-zero) to print using italics. This property affects slave (or Aux) print jobs when the `SlavePrintMode` property is 1 (Graphics mode). This property has no effect when the `SlavePrintMode` property is zero (Text mode).

`PrinterFontName` property (string)
Set this property to the name of the font to use for slave (or Aux) print jobs when the `SlavePrintMode` property is 1 (Graphics mode). This property has no effect when the `SlavePrintMode` property is zero (Text mode).

`PrinterFontSize` property (integer)
Set this property to the size (in points) of the font to use for slave (or Aux) print jobs when the `SlavePrintMode` property is 1 (Graphics mode). This property has no effect when the `SlavePrintMode` property is zero (Text mode).

`PrinterName` property (string)
This is the name of the printer used by the session for slave printer (aux port) output. It may be set to null (“”) to indicate the default printer, or “no printer” to disable slave printing. Otherwise specify the name of the printer as defined in Windows printer settings. *Note: network printers require UNC names.*

`PrinterOrientation` property (integer)
Set this property to 0 use the default page orientation, 1 for portrait or 2 for landscape. This property applies only to slave (or Aux) print jobs and only when the `SlavePrintMode` property is 1 (Graphics mode). This property has no effect when the `SlavePrintMode` property is zero (Text mode).

`PrinterPaperSize` property (integer)
Set this property to 0 use the default paper size for the printer. See the `PrinterPaperSize` property in the `Session` object for a list of paper sizes. This property applies only to slave (or Aux) print jobs and only when the `SlavePrintMode` property is 1 (Graphics mode). This property has no effect when the `SlavePrintMode` property is zero (Text mode).

`PrinterPaperSource` property (integer)
Set this property to 0 use the default paper source for the printer. See the `PrinterPaperSource` property in the `Session` object for a list of paper sources. This property applies only to slave (or Aux) print jobs and only when the `SlavePrintMode` property is 1 (Graphics mode). This property has no effect when the `SlavePrintMode` property is zero (Text mode).

`PrinterTimeout` property (integer)
This property specifies the number of seconds of inactivity before a print job is closed.

`PrintJobEject` property (integer)
This property specifies if a page is ejected before or after a print job. Set this property to 0 for no page ejects, 1 to eject page before print job, 2 to eject page after print job or 3 to eject page before and after print job.

`PrintScreenBackground` property (boolean)
Set to True (non-zero) to print the screen background when using the Print Screen command and the `PrintScreenMode` property is non-zero (Graphics mode); otherwise set to False (zero). This property has no effect when the `ScreenPrintMode` property is zero (Text mode).

`PrintScreenEject` property (integer)
This property specifies if a page is ejected before or after a print screen. Set this property to 0 for no page ejects, 1 to eject page before print screen, 2 to eject page after print screen or 3 to eject page before and after print screen.

`ProtectAttr` property (integer)
Visual attribute number assigned to “protected fields” under Wyse/ADDS emulation. See `Colors()` property for a list of attribute numbers.

`ReadProperties` method
settings*.`ReadProperties` *propbag
This method loads all of the emulator settings from the specified PropertyBag object.

`Save` method
settings*.`Save` *filename
This method saves all of the emulator settings in the specified INI file.

`ScaleFont` property (boolean)
Set to True (non-zero) to enable automatic font scaling; otherwise set to False (zero).

`ScreenPrintMode` property (boolean)
Set to False (zero) for text mode, True (non-zero) for graphics mode. When this property is True, the `PrinterColorMode` and `PrintScreenBackground` properties affect how the screen is printed.

`ScrMode` property (integer)
Set to zero for normal (80 column) mode, 1 for extended (132 column) mode.

`ScrollMode` property (integer)
This controls the appearance of the vertical scroll bar: 0=no scroll bar (scroll bar is only visible if the current number of Rows will not fit in the current window size), 1=always show scroll bar, and 2=automatically show scroll bar when cursor is positioned near right border of window.

`SlavePrintMode` property (integer)
Set to 0 for text mode, 1 for graphics mode. When this property is 1, the PrinterFont…, PrinterOrientation, PrinterPaperSize and PrinterPaperSource properties affect how the slave (Aux) print job is printed. This property can also be set to the ID of any installed "print adapter" script. Print adapter scripts are used to process the print job using an external program, such as PrintWizard. See the Users Guide, Printer Settings, for more information on print adapters. You can easily create your own print adapter scripts for custom print job processing – just use one of the included scripts, such as PrintWizardPCL.atsc, as an example.

`Sound` property (string)
This property may be used to specify a custom sound for the terminal beep. Set this property to null to use the default sound. Set to a valid .wav file name to play the specified wave file. Set to "SystemDefault", "SystemHand", "SystemExclamation" or "SystemAsterisk" to use the sound associated with a system event (as defined in the Control Panel Sounds applet). Set to *frequency,duration* to use a true "beep" at the specified frequency (Hz) and duration (ms).

`SSHAuth` property (integer)
This property may be used to select the authentication method which will be used by a Secrue Shell (ssh) session to authenticate a user to the host. Available authentication methods are public key (2) and password (3). Use zero (0) for the default authentication method (password).

`SSHBreakCharacter` property (integer)
This property specifies the ASCII code of a character which is used to indicate a Break signal to the host. The default value is 3 (**CTRL+C**).

`SSHCipher` property (integer)
This property may be used to select a cipher to be used to encrypt a Secure Shell (ssh) session. Available ciphers are: triple DES (3), blowfish (6), 128 bit AES (7) and 256 bit AES (8). Use zero (0) for the default cipher (triple DES). *AES encryption is only available for SSH2 and requires Windows XP.*

`SSHKeepalive` property (boolean)
If this setting is True (non-zero), AccuTerm will send a special "keepalive" message periodically to maintain the connection (some hosts and routers automatically disconnect when they detect an idle connection).

`SSHKey` property (string)
This property contains the name of the SSH private key file. If no path is included, AccuTerm's default key folder will be used to read the key file.

`SSHNoDelay` property (boolean)
Normally, AccuTerm delays outbound network messages for a short time in order to "coalesce" multiple small messages into a single packet. When this setting is True (non-zero), AccuTerm will send each message as soon as possible. Setting this option may improve throughput, especially when running in AccuTerm's GUI environment and during file transfers, at the expense of increased network traffic.

`SSHVersion` property (integer)
This property may be used to select the desired SSH protocol version. Set this property to zero (0) for automatic protocol selection. Set to 1 for SSH version 1 or set to 2 for SSH version 2. When automatic protocol selection is used, AccuTerm selects the highest protocol version supported by the host.

`StopBits` property (integer)
This is the number of stop bits used by the serial port attached to the session. This property is only meaningful when the communication device type is `atDevSERIAL` or `atDevMODEM`. Acceptable values are 1 or 2. After changing this property, use the `Reset` method for the change to take effect.

`Strip8th` property (boolean)
Set to True (non-zero) to cause AccuTerm to truncate received data to 7 bits; otherwise set to False (zero).

`TelnetAltBreak` property (boolean)
Set to True (non-zero) to send the Telnet Break command when the **BREAK** key is pressed; otherwise set to False (zero) for to send the Telnet Interrupt Process command.

`TelnetBinary` property (boolean)
Set to True (non-zero) to enable Telnet binary communication mode; otherwise set to False (zero) for text communication mode.

`TelnetBypass` property (boolean)
Set to True (non-zero) to bypass initial option negotiation; otherwise set to False (zero) for normal negotiation. This is required for certain hosts like Pick's D3 which do not implement the complete Telnet protocol.

`TelnetEcho` property (boolean)
Set to True (non-zero) to enable Telnet remote-echo mode; otherwise set to False (zero) for local echo mode. Note: when using local echo, you should also set the `Duplex` property to "half".

`TelnetKeepalive` property (boolean)
If this setting is True (non-zero), AccuTerm will send a special "keepalive" message periodically to maintain the connection (some hosts and routers automatically disconnect when they detect an idle connection). *Note: this setting should not be used when connecting to a D3/NT host.*

`TelnetNoDelay` property (boolean)
Normally, AccuTerm delays outbound network messages for a short time in order to "coalesce" multiple small messages into a single packet. When this setting is True (non-zero), AccuTerm will send each message as soon as possible. Setting this option may improve throughput, especially when running in AccuTerm's GUI environment and during file transfers, at the expense of increased network traffic.

`TermType` property (integer)
This is the terminal emulation setting for the session. Possible values are `atTermTTY` (basic TTY), `atTermVPA2` (ADDS Viewpoint A2), `atTermVP60` (ADDS Viewpoint 60), `atTermP60` (ProComm Viewpoint 60), `atTermA2E` (ADDS Viewpoint A2 Enhanced), `atTermWY50` (Wyse 50), `atTermWY60` (Wyse 60), `atTermVT52` (DEC VT52), `atTermVT100 (DEC VT100)`, `atTermVT220` (DEC VT220), `atTermVT320 (DEC VT320)`, `atTermVT420 (DEC VT420)`, `atTermLinux (Linux Console)`, `atTermSCO (SCO Console)`, `atTermANSI` (ANSI BBS), `atTermPICKMON` (Pick PC Monitor), `atTermP30` (MDIS P30) or `atTermTEK` (Tektronix 4014).

`U2DeviceLicensing` property (boolean)
If this setting is True (non-zero), multiple connections (sessions) to Enterprise versions of UniData and UniVerse consume only a single user license. Contact your IBM dealer for information regarding device licensing.

`WriteProperties` method
settings*.`WriteProperties` *propbag
This method saves all of the emulator settings into the specified PropertyBag object.

`XmodemTimeout` property (integer)
Specifies the number of seconds before Xmodem file transfer operations time out.

`YmodemTimeout` property (integer)
Specifies the number of seconds before Ymodem file transfer operations time out.

`ZmodemTimeout` property (integer)
Specifies the number of seconds before Zmodem file transfer operations time out.

`ZmodemAuto` property (boolean)
Set to True (non-zero) to enable automatic Zmodem downloads, otherwise set to False (zero).

The ScreenBlock Object
The `ScreenBlock` object is created by the `Session GetBlock` method, and is used to preserve and restore the contents of a portion of the terminal screen. The `Session SetBlock` method restores the contents of the `ScreenBlock` object back onto the terminal screen.

The Menu Object
Users interact with AccuTerm using menus, toolbars, status bars and the session bar. These objects, called "bands" (object type is `MnuBand`) contain "tools" (object type is `MnuTool`) which perform actions or display status information. The AccuTerm `Menu` object contains bands, which in turn, contain tools.

Both the `AccuTerm` object and each `Session` object contain a `Menu` object. The `Menu` object is the root object of AccuTerm's menu structure. When no sessions are open, the menu object of the AccuTerm object is the operational menu. When any session is open, the menu object of the active session is the operational menu.

You can use the `Menu` object (more specifically the `MnuBand` and `MnuTool` objects contained within the `Menu` object) to customize the operation of the menu, toolbar, and status bar, however, you need to use the menu designer to add or remove bands or tools from the menu.

Typically, you only need access to the tool objects, and the default method of the `Menu` object, Item, returns a `MnuTool` object reference given the tool's ID. See the online help, **AccuTerm constants**, for a list of menu IDs.

`Count` property (integer)
This read-only property is the number of `MnuBand` objects contained in the menu.

`MnuBands` property (array of `MnuBand` objects)
This read-only property returns a reference to a `MnuBand` object given the band name or one-based band index.

`Item` method (default)
This method returns a reference to a `MnuTool` object corresponding to the specified tool ID. This is the default method of the `Menu` object, and as such, the method name may be omitted. For example, to disable the "close" menu tool, use:

```
AccuTerm.Menu(FILE_CLOSE).Enabled=False
```

The MnuBand Object
A `Menu` object contains zero or more `MnuBand` objects. Examples of `MnuBand` objects are the main menu, context (right-click) menu, toolbar, and status bar. A `MnuBand` object contains MnuTool objects.

`BandType` property (integer)
This read-only property is the type of band: 0 = toolbar, 1 = main menu, 2 = popup menu, 3 = status bar, 4 = session bar.

`Caption` property (string)
This property is used for floating toolbars, and is the caption displayed in the floating window title.

`Count` property (integer)
This read-only property is the number of `MnuTool` objects contained in the band.

`DockingArea` property (integer)
This property specifies the position of the toolbar or status bar: 1 = top, 2 = bottom, 4 = left, 8 = right, 16 = floating.

`MnuTools` property (array of `MnuTool` objects)
This read-only property returns a reference to a `MnuTool` object given the tool ID or one-based tool index.

`Name` property (string)
This read-only property is the name of the `MnuBand` object.

`Visible` property (string)
This property is True (non-zero) if the band is visible.

The MnuTool Object
The `MnuTool` object is at the lowest level of the `Menu` object hierarchy, and contains properties for the caption, state and action of a menu item, toolbar button or status bar field.

`Action` property (string)
This property contains the action to perform when the menu item or button is clicked. All built-in menu items have a default action associated with them, which is the tool ID prefixed by a pound sign ("#") and enclosed in square brackets (**[ ]**). To execute a macro (instead of the default action), enclose the macro statement in square brackets (**[ ]**). To transmit a character sequence to the host, set the action property to the desired sequence using the same syntax as programmed function keys.

`Caption` property (string)
This property is the caption displayed for the menu or status bar item. The caption is not displayed in a toolbar button.

`Checked` property (integer)
Set this property to True to display a checked box next to the caption in popup menus or sub-menus.

`Enabled` property (integer)
Set this property to True to enable a menu item. If this property is False, the item is disabled and is displayed in a gray color.

`ToolID` property (long)
This read-only property is the tool ID.

`ToolTipText` property (string)
This property is the text which is displayed in a floating "hint" window when the cursor is paused over a toolbar button or status bar panel.

`Visible` property (string)
This property is True if the tool (menu item, toolbar button or status bar panel) is visible.

CUSTOMIZING THE MENU AND TOOLBAR

AccuTerm 2K2 includes a menu design function which can be used to customize AccuTerm's main menu, context menu, toolbar and status line. *Use caution when modifying AccuTerm's menu structure; it is possible to make AccuTerm non-functional.* The menu designer allows you to add, remove and modify menu items, sub-menus, toolbar buttons and status line fields.

Using the Menu Designer

To start the Menu Designer, select **Customize Menu...** from the **Tools** menu.

The initial panel in the designer displays the current menu file name, and the scope of the current menu. You can alter the scope by selecting one of the other scope options. You can also remove a custom menu from the file by clicking on one of the **Remove** check-boxes and clicking the **Remove selected menu** button.

Menu Scope
AccuTerm 2K2 supports menus at three different levels: Session, User and Application:

> **Session**
> A Session level menu is stored in the session configuration file. Any time that the session file is activated, AccuTerm uses the menu for that session. A Session level menu overrides a User level or an Application level menu. *Note: the Session scope is only available if you started the Menu Designer from an active session whose session configuration file has already been saved; if the Session option is not enabled, exit the designer and save your session file, then open the designer again.*
>
> **User**
> A User level menu is stored in the current user's profile (usually under Documents and Settings\user\Local Settings\Application Data\...). Whenever the current user opens AccuTerm, or any

session that does not have a Session level menu, AccuTerm uses the User level menu. A User level menu overrides an Application level menu, but not a Session level menu.

Application
An Application level menu is stored in AccuTerm's main program directory, and, if present, is used by AccuTerm whenever there is no User level menu and no Session level menu. The Application level menu can only be created or modified by the system administrator. *If the Application option is disabled it means that you do not have the required administrator privledges to modify the menu.*

You must select the menu scope before you can customize the menu. Once you have selected the scope, click the **Continue** button to display the selected menu structure.

Menu Structure
The AccuTerm2K2 menu is a hierarchical structure of *Bands* and *Tools*. A *band* is a collection of related *tools* organized in a logical manner. There are different types of *bands* (menu, toolbar, status bar), as well as different types of *tools* (menu item, toggle item, panel, etc). *Bands* contain *tools* which perform *actions*.

After selecting the menu file (scope), the Menu Designer displays the menu structure using a tree on the left half of the designer window. The root node in the tree is **AccuTerm Menu & Toolbar**. Under the root are nodes for each *band* in the menu structure. Each menu, toolbar, status bar and session bar is considered a *band*. There can be only one **Main Menu** *band* and only one **Session Bar** *band*. It is possible to have multiple **Toolbar**, **Status Bar**, and **Popup Menu** *bands*, although having multiples may be confusing to the user.

Whenever a node (*band* or *tool*) in the menu tree structure is selected, the right half of the designer window displays the *properties* of that node. You customize the selected object by adjusting its *properties*.

When you are finished customizing the menu, click the **OK** button. To discard any changes, click the **Cancel** button.

Adding bands and tools to the menu
Use the **New** button below the menu tree to add a new item to the selected node. If the selected node is the root node, clicking the **New** button will display a popup menu showing the types of *bands* that you can add: **Popup Menu**, **Toolbar** or **Status Bar**.

If the selected node is a *band*, clicking the **New** button will display a popup menu showing the types of *tools* that you can add to the *band*.

If the selected node is a *sub-menu tool* (an intermediate node in a main menu or popup menu or a dropdown toolbar button), clicking the **New** button will display a popup menu showing the types of *tools* that can be added to the sub-menu or dropdown button.

You can also right-click on any node in the menu tree and select **New** from the popup menu to add an item to the node.

Removing bands and tools from the menu
Use the **Delete** button to remove the selected node from the menu tree. You can also right-click on any node in the menu tree and select **Delete** from the popup menu to remove the node.

Repositioning tools in the menu
You can reposition tools in the menu by dragging and dropping, or by using the **Cut**, **Copy** and **Paste** options from the right-click popup menu. If you hold the **SHIFT** key while dragging, the tool will be copied; without the **SHIFT** key, the tool will be moved.

Restoring the default menu structure
Click the **Default** button to restore the menu structure to its original, default structure.

Band Properties

Name

This is the name of the band. Each band must havea unique name. Default band names are “Menu”, “Toolbar”, “StatusBar”, “ContextMenu” and “SessionBar”.

Type

The **Type** field shows the selected band type: **Main Menu**, **Popup Menu**, **Toolbar**, **Status Bar** or **Session Bar**. You cannot change the band type; the type is determined when you create a new band using the **New** button.

Style

The **Style** field is used to select the band style. Usually this is “Normal”, but you can choose one of the other styles if you want to display icons in the **Sesion Bar** or **Status Bar**, or display captions in the **Toolbar**.

Dock Position

You can change the **Dock position** of **Toolbar** and **Status Bar** bands. Normally, **Toolbar** bands are docked at the *top* of the window, and **Status Bar** bands are docked at the *bottom.*

Enabled

When a band is *enabled,* the user can click on tools in that band. If it is *disabled,* tools on the band do not respond to user clicks or keyboard shortcuts.

Visible

If you want the user to see the band, set the **Visible** property. If you clear this property, the band will be hidden.

Large Icons

The **Large icons** property is set at runtime by the AccuTerm application. Do not alter this property in the designer.

Auto Wrap

If you check the **Auto wrap** property for a **Toolbar** band, and the window is not wide enough to display all of the **Toolbar** buttons, the **Toolbar** will “wrap” at the right edge of the window and show on two or more lines.

Tool Properties

ID

The **ID** property identifies each tool. Many ID's are built-in and can be selected from the dropdown list. When you create a new tool that does not duplicate the function of a built-in tool, you must specify a unique **ID** in the range of 10000 to 19999.

Type

The tool **Type** depends on the type of band the tool is on:

Menu or Popup Menu

Menu Item: when the user clicks a Menu Item, an action is performed.

Sub-Menu: when the user clicks a Sub-Menu, another menu level is displayed.

Toggle Item: when the user clicks a Toggle Item, the item changes state (checked vs. unchedked). When a Toggle Item is "checked", it may display a different icon, or a checkmark if no icon has been designated. When a Toggle Item changes state, its action is performed.

Group Item: a Group Item is like a Toggle Item, except that only one item in the group can be "checked" at a time. A group is delimited by a menu item that has the **Begin group** option checked.

Toolbar

Normal Button: when the user clicks a Normal Button, an action is performed.

Dropdown Button: when the user clicks a Dropdown Button, another popup menu (sub-menu) is displayed.

Toggle Button: when the user clicks a Toggle Button, the button changes state (depressed vs. raised). When a Toggle Button is "checked", it may display a different icon from the "unchecked" state. When a Toggle Button changes state, its action is performed.

Group Button: a Group Item is like a Toggle Button, except that only one button in the group can be "checked" at a time. A

group is delimited by a button item that has the **Begin group** option checked.

Status Bar

Normal panel: displays text and/or icon in the Status Bar.

Num Lock panel: displays the state of the Num Lock key in the Status Bar.

Caps Lock panel: displays the state of the Caps Lock key in the Status Bar.

Scrl Lock panel: displays the state of the Scroll Lock key in the Status Bar.

Time panel: displays the current time in the Status Bar.

Date panel: displays the current date in the Status Bar.

Date/Time panel: displays the current time and date in the Status Bar.

Session Bar

A Session Bar does not have any design-time tools. The Session Bar buttons are created automatically.

Style
The tool **Style** determines whether text, icon or both are displayed. You can also choose "normal", which will display the most common style for the tool and band type.

Caption
The tool **Caption** text is displayed in **Menu** and **Status Bar** tools. It can optionally be displayed for a **Toolbar** button. For **Menu** tools, you can create an "accelerator", or "hot key", for the item by inserting an ampersand (**&**) before the letter you want to use for the accelerator in the **Caption**.

Alignment
For **Status Bar** tools, choose the text alignment: *left, right* or *center.*

Border Style
For **Status Bar** tools, choose the panel border style: *inset, raised* or *none.*

Panel Size
For **Status Bar** tools, choose the panel size: *fixed width*, *fit contents* or *spring*. If you choose *fixed width*, and set the **Minimum Panel Width** to zero, the panel will be sized using the text in the **Caption** property. If you choose *fit contents*, the panel will adjust size any time the **Caption** text is updated. If you choose *spring*, the panel size is like *fit contents*, but varies with window size. All *spring* panels are resized when the window size changes.

Minimum Panel Width
Specify the size of a *fixed width* panel, or the minimum size for a *fit contents* or *spring* panel. This value is in pixels.

Shortcut
For **Main Menu** tools only, you can select a keyboard shortcut for that tool. This shortcut is usually combined with **Ctrl**, **Shift** and **Alt** modifier keys. Note that the shortcut overrides any other use of the key. For example, if you designante **CTRL+C** as a shortcut, AccuTerm will be unable to send the **CTRL+C** control code when you press **CTRL+C**.

Tool Tip
Tool tip text is displayed when you leave the mouse over a **Toolbar** tool or **Status Bar** panel.

Action
The **Action** field determines what happens when the user clicks the tool. For built-in tools, the **Action** is the same as the **ID**. For user-defined tools, you can either enter a string to send to the host, or a script enclosed in square brackets (**[ ]**), in this field. The syntax is exactly the same as function key programming (see the Users Guide for details).

Enabled
When a tool is *enabled*, the user can click on it. If it is *disabled*, the click is ignored. For built-in tools, AccuTerm will adjust this property depending on the program and session state.

Visible
If you want the user to see the tool, set the **Visible** property. If you clear this property, the tool will be hidden. For built-in tools,

AccuTerm will adjust this property depending on the program and session state.

Checked
This is the initial state of a **Toggle Item**, **Group Item**, **Toggle Button** or **Group Button**. For built-in tools, AccuTerm will adjust this property depending on the program and session state.

Begin Group
If the **Begin group** property is set, there will be a separator before the menu item or toolbar button. This property is also used to identify which buttons belong to a group of **Group Items** or **Group Buttons**.

Select Image…
Click the **Select image** button to open the **Image Selection** dialog.

Image Selection
When you click the **Select image** button in the Tool properties panel, the **Image Selection** dialog is displayed. Use this dialog to select the icon to display for the tool. You can select both normal (un-checked) and checked icons. For each tool, the image can be one of the built-in icons, or you can specify a file name for the image to be loaded from at run time.

Image Source
Choose the **Image source**, or specify that the tool does not have an icon.

Normal Image
Check this box if the tool has a normal (un-checked) icon. Enter the image file name (if the tool's icon is loaded from a file at runtime), or the image ID if the tool uses a built-in image. Click the **Browse** button to browse for image files or built-in images.

Chheckd Image
Check this box if the tool has a checked icon that is displayed when the tool is in its "checked" state. Enter the image file name (if the tool's icon is loaded from a file at runtime), or the image ID if the tool uses a built-in image. Click the **Browse** button to browse for image files or built-in images.

ACCUTERM PROGRAMMING

AccuTerm 2K2 contains several "private" commands which may be sent by host application programs. These commands allow for remote control of file transfer and data capture, executing Windows programs or DOS commands, enabling the mouse, displaying images, and programming the function and keypad keys. These commands are valid from any terminal emulation, and are described below:

ESC STX < command CR

Executes the Windows or DOS command ***command***. If ***command*** specifies a file or URL, the file or URL is opened using the default application defined for that file. AccuTerm returns to emulation mode immediately. ***Command*** is executed concurrently with the terminal session.

ESC STX > command CR

Executes the Windows or DOS command ***command***. If ***command*** specifies a file or URL, the file or URL is opened using the default application defined for that file. The terminal session is suspended until the command completes or the document is closed.

ESC STX **0** Disable mouse input; turns off mouse cursor.

ESC STX **1** Enable mouse input; turns on mouse cursor. Transmits mouse location whenever a mouse button is pressed. The format of the mouse location depends on terminal emulation:

ASCII: ***STX b CR ccc . rr CR***
ANSI 7 bit: ***ESC [n ~ ESC [r ; c R***
ANSI 8 bit: ***CSI n ~ CSI r ; c R***

ASCII: ***b*** indicates which mouse button was pressed (**p**=left, **q**=right, **r**= center single click, **P**=left, **Q**=right, **R**=center double click), ***ccc*** is the three digit column of the mouse cursor and ***rr*** is the two digit row of the mouse cursor (both in decimal, **000.00** is the upper left corner). If ***b*** is lower case, the mouse was clicked once, ***b*** is uppercase if double clicked.

ANSI: ***n*** indicates which mouse button was pressed (**101**=left, **102**=right, **103**=center single click, **111**=left,

112=right, **113**=center double click), ***r*** is the row of the mouse cursor and ***c*** is the column of the mouse cursor (both in decimal, where **1;1** is the upper left corner).

Note: if the mouse is clicked on an image that is displayed on the terminal screen, the location in the above responses is replaced by the image file name.

ESC STX **2** Enable mouse input in SystemBuilder compatibility mode; turns on mouse cursor. Transmits mouse location whenever a mouse button is pressed:

STX ~ CR b ; c ; r CR

b indicates which mouse button was pressed (1 = left, 2 = right, 4 = center, ***c*** is the column of the mouse cursor and ***r*** is the row of the mouse cursor (both in decimal, **0;0** is the upper left corner).

ESC STX **D** ***p o m ; path CR***

Download file from host to PC. Protocol ***p*** may be **A** (ASCII), **K** (Kermit), **X** (Xmodem), **Y** (Ymodem) or **Z** (Zmodem); Overwrite ***o*** may be **O** (overwrite) or **N** (no overwrite) or, if protocol is **Z**, overwrite may be **R** (resume); Mode ***m*** may be **T** (text) or **B** (binary). ***Path*** is the drive, directory and file name of the file being received. When using Kermit, Ymodem or Zmodem protocols, only drive and directory need be specified, as the file name is included in the transfer protocol; however if the file name is specified here, it overrides the file name included in the transfer protocol.

ESC STX **U** ***p m ; path CR***

Upload file from PC to host. Protocol ***p*** may be **A** (ASCII), **K** (Kermit), **X** (Xmodem), **Y** (Ymodem) or **Z** (Zmodem); Mode ***m*** may be **T** (text) or **B** (binary). ***Path*** is the drive, directory and file name of the file to send to the host. Wildcard characters (* or ?) are valid when using Kermit, Ymodem or Zmodem protocols.

ESC STX **S** Returns status of last file transfer. Status message is:

Status: *s* **files** *f* **bytes** *b* ***CR***

where ***f*** is the number of files transferred, ***b*** is the number of bytes transferred, and ***s*** is the transfer status:

0 = transfer successful
1 = unable to open file
2 = transfer aborted by operator
3 = file already exists
4 = terminated due to timeout
5 = terminated due to corrupted data
6 = invalid packet type
7 = terminated by remote program
8 = 8 data bits required for protocol
9 = software flow control not allowed for protocol
10 = the destination file is write protected

ESC STX **C** ***o s f*** **;** ***path CR***

Begin capture. Mode ***o*** may be **O** (overwrite), **A** (append), **N** (new file only) or **C** (clipboard). Source ***s*** may be **R** to capture received data, or **P** to capture printed data (data received while the slave port is on). Filter ***f*** may be **T** to capture text only (strips control characters) or null to capture unfiltered data. ***Path*** is the drive, directory and file name where the captured data is to be stored. All characters received (or printed) are stored in the file (or clipboard) until capturing is disabled (via local or remote command).

Note: when capturing to the clipboard, ***path*** is ignored.

ESC STX **C X**

End capture. The file containing the captured data is closed.

ESC STX **I** Returns AccuTerm release, serial number, type and licensee information. Message format is:

ACCUTERM/WIN ***rel serial type licensee... CR***

where ***rel*** is the AccuTerm release number, ***serial*** is the program serial number, ***type*** is **SINGLE**, **SITE**, **CORP**, **DEALER** or **DEMO**, and ***licensee*** is the name under which the program has been licensed.

ESC STX ? Returns a string indicating the platform, product type, license type, capabilities and automation services. Message format is:

platform * product * license * capabilities * services CR

where ***platform*** is 3 (Win32); ***product*** is 4 = AccuTerm (standard version), 5 = AccuTerm Internet Edition, 6 = AccuTerm Emulator ActiveX Control, 7 = AccuTerm Lite; release number, ***serial*** is the program serial number, ***license*** is 1 = single user, 2 = site, 3 = enterprise, 5 = developer, 7 = internet, 8 = component, 9 = evaluation; ***capabilities*** is a string of letters indicating the various capabilities where:

A = ASCII protocol supported
B = Border effects (visual styles) enabled
C = Capture supported
D = Download supported
E = Execute command supported
G = GUI supported
H = Server mode supported
I = Image display supported
J = Screen save & restore supported
K = Kermit protocol supported
O = Object Bridge supported
P = Packetized messages supported
Q = Message integrity checks supported
R = Reliable connection
S = Scripting supported
T = File transfer error info supported
U = Upload supported
V = Host capabilities command supported
X = File conversion supported

services is a string of letters indicating which automation services are available (a = Object Bridge, b = file converter, g = GUI).

ESC STX = host-capabilities * buffer-size CR

Informs AccuTerm of any special host program capabilities. ***host-capabilities*** is a string of letters indicating various host features:

D = host supports tag notation for system delimiters
C = host supports message checksums
M = host supports message length check

buffer-size is optional, and specifies the maximum packet size that the client can send to host

ESC STX **%** **0**
Returns the AccuTerm program directory, followed by a ***CR***.

ESC STX **X** Terminates AccuTerm session. If only one session exists, then AccuTerm is terminated also.

ESC STX **W** Saves the current session settings to disk. If no session file name exists, then one will be prompted for.

ESC STX **E** Selects extended (132 column) video mode as defined by the extended columns and rows in the **Screen Settings** category in the **Settings** dialog box.

ESC STX **N** Selects normal (80 column) video mode as defined by the normal columns and rows in the **Screen Settings** category in the **Settings** dialog box.

ESC STX **L** Sets the keyboard **CapsLock** state.

ESC STX **M** Clears the keyboard **CapsLock** state.

ESC STX **Y** ***path CR***
Send clipboard or file to host. If ***path*** is null, the contents of the clipboard are sent, otherwise the specified file is sent. Each line is terminated as specified in the ASCII settings (**File Transfer Settings**). A ***SUB*** (**CTRL-Z**) is sent after the last line.

ESC STX **m** ***filename CR***
Plays MIDI sound file ***filename***. The file does not need to be a local file; if an internet connection is available the filename can specify a URL instead of a local or network file.

ESC STX **w** ***filename CR***
Plays Wave sound file ***filename***. The file does not need to be a local file; if an internet connection is available the filename can specify a URL instead of a local or network file.

ESC STX **P** ***command CR***

Executes macro command ***command***. See section on **"Scripting"** for details on the available commands. Multiple statements may be executed; separate each statement with ***LF*** or ***EM*** control characters. Note: you can call subroutines or functions contained within the script loaded in the main script window. For example, to call subroutine FOO, simply send: ***ESC STX*** **P FOO** ***CR*** . (Note: spaces are not part of the sequence, they are only shown here for clarity).

ESC STX **F** ***t s d k data CR***

Program function and keypad keys. Type ***t*** may be **N** (normal function keys), **C** (control function keys), **A** (ALT function keys) or **K** (keypad keys). Shift ***s*** may be **U** (unshifted) or **S** (shifted). Destination ***d*** may be null (program key) or **T** (program function bar button caption). Key code ***k*** may be digits **0** to **9** or **: ; < = >** according to the following table. ***Data*** contains the function key sequence data. Control codes may be represented by prefixing the a letter or symbol with **^** (e.g. to program a carriage return, enter **^M**). To program a **^**, enter **^^**. To program a **CTRL+^**, enter **^~**. Note: if ***data*** is enclosed in brackets (**[ ]**), then when the key is pressed, it will be interpreted as a VBA script statement, rather than sent to the host.

Key Code	Function key	Editing key
0	F1	BKSP
1	F2	TAB
2	F3	INS
3	F4	DEL
4	F5	HOME
5	F6	END
6	F7	PGUP
7	F8	PGDN
8	F9	LEFT
9	F10	RIGHT
:	F11	UP
;	F12	DOWN
<		ESC
=		ENTER
>		KPD ENTER

For example, to program the **END** key to send the word "END", followed by a carriage return, the following sequence would be used:

ESC STX **F K U 5 E N D ^ M** ***CR***

ESC STX **iL** **,** ***filename , col , row , width , height , aspect, border CR***

Displays the image file ***filename*** at column ***col*** and row ***row***. ***Height*** and ***width*** are optional; if specified (and not zero), the image is scaled to ***height*** rows and ***width*** columns. Otherwise, the original image size is used. If ***aspect*** is non-zero, the image aspect ratio is preserved (the specified width or height is reduced to preserve the aspect ration). ***Border*** is **N** for no border, **B** for simple border, **R** for raised border or **I** for inset border style. Image file types supported include bitmap, Windows metafile, JPEG, GIF, TIFF and TARGA. Images are considered "protected" data; a "clear un-protected characters" command will not clear images from the terminal screen.

The image file does not need to be a local file; if an Internet connection is available the filename can specify a URL instead of a local or network file.

ESC STX **iD** **,** ***filename CR***

Removes the displayed image file ***filename*** from the screen.

ESC STX **iC** ***CR***

Clears all displayed images from the screen.

ESC STX **h** ***CR***

Resets the mouse pattern table.

ESC STX **h** ***button HT pattern HT click HT dblclk CR***

Adds an entry to the mouse pattern table.

ESC STX **A** ***color*** ***CR***

Sets the foreground (text) color to ***color***.

ESC STX **B** ***color*** ***CR***

Sets the background color to ***color***.

Color	Display
0	Black
1	Dark Blue
2	Dark Green
3	Turquoise
4	Dark Red
5	Purple
6	Olive
7	Light Gray
8	Dark Grey
9	Blue
10	Green
11	Cyan
12	Red
13	Magenta
14	Yellow
15	White

Note: the actual display color may vary from the color shown in the table, since it is possible to modify the palette, either using the Color Settings section of the Settings dialog, or from code.

ESC STX **r** ***x1 , y1 , x2 , y2 , fill , border*** ***CR***

Draw rectangle. ***x1*** , ***y1*** is the upper-left corner and ***x2***, ***y2*** is the lower-right corner. Coordinates are in character columns and rows. The upper-left corner of the screen is (0, 0).

The rectangle can be filled using either a visual attribute or a background color. To fill with a visual attribute, ***fill*** is **A0 … A63**. The attribute number is formed by adding the following values:

0 = normal
1 = invisible
2 = blinking
4 = reverse
8 = underline

16 = dim
32 = bright

To fill with a background color (as described above, ***fill*** is **B0 … B15**. To draw a border without filling the rectangle, fill is **N**.

A border can be drawn around the rectangle. ***Border*** is **N** (no border), **B** (flat border), **I** (inset border) or **R** (raised border).

ESC STX **jS , *ID , col , row , width , height , page*** ***CR***

Saves a copy of the specified screen block (text, colors and visual effects) and terminal state in memory and associates the block with the specified identifier (***ID***) which can be an arbitrary alpha-numeric string (may not contain commas). Any number of screen blocks can be saved. ***Col*** and ***row*** specify the upper-left corner of the block. If either is omitted or null, zero is assumed. ***Width*** and ***height*** specify the size of the block. If ***width*** or ***height*** is omitted or null, the screen width or height is used. ***Page*** optionally specifies the terminal page for the block, and if omitted or null, the current page is used.

ESC STX **jR , *ID , col , row , page , state*** ***CR***

Restores the screen block associated with the specified identifier (***ID***) to the screen. ***Col*** and ***row*** specify the upper-left corner of the destination. If either is omitted or null, the original position of the saved block is assumed. ***Page*** optionally specifies the terminal page for the destination, and if omitted or null, the current page is used. If ***state*** is 1, the terminal state (current attribute, cursor position, write protect mode, etc.) is restored along with the screen contents.

ESC STX **jD , *ID*** ***CR***

Deletes the screen block associated with the specified identifier (***ID***) from memory.

WYSE PROGRAMMING

This section describes the command sequences for programming the Wyse 50, Wyse 60 and ADDS Viewpoint Enhanced terminal emulations. These three emulations use a common command set with a few differences. The differences are noted.

Operating Modes

ESC ` *n* Set mode ***n***. This command is used to set many of the terminal's operating modes. The mode ***n*** values and their function are shown in the table below:

n	Function
0	Cursor off
1	Cursor on
2	Block cursor
3	Line cursor
A	Normal protect character
6	Reverse protect character
7	Dim protect character
8	Screen off
9	Screen on
:	80 column screen
;	132 column screen
B	Protect blink on
C	Protect invisible on
E	Protect underline on
F	Protect reverse on
G	Protect dim on

ESC B Places terminal in local mode.

ESC C or **ESC D F**
Places terminal in full duplex mode.

ESC **D H** Places terminal in half duplex mode.

ESC **N** Disables auto scrolling. Normally if the cursor is moved down below the last line, the screen is scrolled up one line. If auto scrolling is disabled, the cursor moves to the top line and no scrolling takes place.

ESC **O** Enables auto scrolling.

ESC **X** or ***ESC*** **u**
Turns the monitor mode off.

ESC **U** Turns the monitor mode on.

ESC **k** Turns the local edit submode on. This command is not supported by AccuTerm and is ignored.†

ESC **l** Turns the local edit submode off. This command is not supported by AccuTerm and is ignored.

ESC **q** Turns the insert mode on. In this mode, all characters sent to the screen are inserted into the line with any existing characters moved one column to the right.

ESC **r** Turns the insert mode off.

ESC **#** or ***SI***
Lock keyboard.†

ESC **5** or ***EOT***
Lock keyboard.‡

ESC **"** or ***SO***
Unlock keyboard.†

ESC **6** or ***STX***
Unlock keyboard.‡

GS If the graphics mode is enabled, this command puts the terminal in the Tektronix emulation mode. If the graphics mode is disabled, this command is ignored.

† Only valid in Wyse 50 and Wyse 60 modes.
‡ Only valid in Viewpoint Enhanced mode.

ESC* 1 *ESC FF
If the graphics mode is enabled, this command puts the terminal in the Tektronix emulation mode. If graphics mode is disabled, it sets a tab stop at the current cursor column.

***ESC* % ! 0** Enters Tektronix 4100 graphics mode.

CAN Exits from the Tektronix graphics emulation mode.

CAN Turns on the cursor.‡

ETB Turns off the cursor.‡

***ESC* 2** Exits from the Tektronix graphics emulation mode. If the terminal is not in the graphics emulation mode, it clears any tab stops at the current cursor column.

ESC* F *message CR
Displays a message in the host message line. The message string can be up to 46 characters in 80 column mode and 98 characters in the 132 column mode.

ESC* c ; *message EM
Programs the answerback message (up to 30 characters).

ESC* c <** Sends the answerback message to the host, followed by ***ACK. If no answerback message has been programmed, simply sends ***ACK***.

***ESC* c =** Erases the answerback message.

***ESC* d /** End of line wrap on. When the cursor reaches the end of a line, it will wrap to the beginning of the next line.

***ESC* d .** End of line wrap off. When the cursor reaches the end of a line, it does not wrap to the beginning of the next line.

***ESC* e &** Set **CapsLock** keyboard state.

***ESC* e '** Clear **CapsLock** keyboard state.

‡ Only valid in Viewpoint Enhanced mode.

ESC e . Do not clear screen when screen size changes.

ESC e / Clear screen when screen size changes.

ESC ^ 0 Normal display (light characters, dark background).

ESC ^ 1 Reverse display (dark characters, light background).

ESC ~ SPACE
Enhance mode off. If the current emulation is ADDS VPA2E, then the emulation will change to ADDS VPA2.

ESC ~ ! Enhance mode on. If the current emulation is ADDS VPA2, then the emulation will change to ADDS VPA2E.

ESC ~ " Select Wyse 50 emulation.

ESC ~ % Select ADDS VPA2E emulation.

ESC ~ 4 Select Wyse 60 emulation.

ESC ~ 6 Select VT-52 emulation.

ESC ~ 8 Select Viewpoint 60 emulation.

ESC ~ ; Select VT-100 emulation.

ESC ~ < Select VT-220 7-bit emulation.

ESC ~ = or **ESC ~ A**
Select VT-220 8-bit emulation.

ESC ~ B Select VT-320 7-bit emulation.

ESC ~ C Select VT-320 8-bit emulation.

ESC ~ > Select Tektronix 4014 emulation.

ESC C V or **ESC C W**
Save current session settings to disk. If no session file name exists, then one will be prompted for.

ESC C X Reset (reload session configuration file).

Character Set Selection

Wyse terminals have two character sets (primary and secondary) of 128 symbols each, and four font banks of 128 symbols. Each font bank can be assigned a pre-defined symbol set (like PC Multinational), and each character set can be assigned to one font bank.

By default, the native Wyse 50 symbol set is assigned to font banks 0, 2 and 3. The PC Multinational symbol set is assigned to font bank 1. Font bank 0 is assigned to the primary character set and font bank 1 is assigned to the secondary character set.

ESC C D Selects the primary character set. This is the default, and when selected, the primary character set is displayed for character codes 0 to 127, and the secondary character set is displayed for codes 128 to 255.

ESC C E Selects the secondary character set. When the secondary character set is selected, it is displayed for character codes 0 to 127 and again for codes 128 to 255.

ESC C B *bank*
Assigns font bank ***bank*** (0, 1, 2 or 3) to the primary character set.

ESC C C *bank*
Assigns font bank ***bank*** (0, 1, 2 or 3) to the secondary character set.

ESC C @ *bank set*
Assigns pre-defined symbol set ***set*** to font bank ***bank. Set*** can be one of the following:

set	**symbol set**
@ or **`**	Native Wyse 50
A or **a**	PC Multinational
B , **c** , **d** , **G** , **H** or **g**	US ASCII
J , **j** , **N** , **I** or **e**	DEC Graphics / ISO Latin-1
D or **b**	PC Standard

Cursor Positioning

FF Cursor right. If the cursor is on the last column of the line, the cursor will wrap to the next line, possibly scrolling the screen up.†

ACK Cursor right. If the cursor is on the last column of the line, the cursor will wrap to the next line, possibly scrolling the screen up.‡

BS Cursor left. If the cursor is at the beginning of the line, it is moved up to the last column of the previous line. If the cursor is at the home position, it is moved to the lower right hand corner of the screen.

NAK Cursor left. Same as the ***BS*** command.‡

HT or ***ESC*** **i**
Moves the cursor to the next programmed tab stop. For this command to work, tab stops must be programmed with the ***ESC*** **1** command.

ESC **I** Move the cursor left to the previous tab stop.

LF Cursor down. If the cursor is on the last line of the screen and the "no scroll" mode is turned off, the screen will scroll up one line. Otherwise, the cursor will move to the top line of the screen.

VT Cursor up. If the cursor is at the top row, it is moved to the bottom row.†

SUB Cursor up. If the cursor is at the top row, it is moved to the bottom row.‡

† Only valid in Wyse 50 and Wyse 60 modes.
‡ Only valid in Viewpoint Enhanced mode.

ESC **j** Cursor up. If the cursor is at the top row, the screen is scrolled down.

CR Moves the cursor to the first column (column zero) of the current row.

US Moves the cursor down one row and over to the first column (column zero).

DEL Ignored.

RS or ***ESC*** **{**
Moves the cursor to the home position (upper left corner of the screen).

VT r Moves cursor to row ***r***, where ***r*** is a valid row code from the Viewpoint Cursor Address Table (Appendix B).‡

DLE c Moves the cursor to column ***c*** where ***c*** is a valid column in the from the Viewpoint Cursor Address Table (Appendix B).‡

ESC = r c Moves the cursor to row ***r*** and column ***c***. ***r*** and ***c*** are single byte cursor address codes from the Wyse Cursor Address Table (Appendix A). Note that this command cannot position the cursor past column 95.

ESC **Y** ***r c*** Moves the cursor to row ***r*** and column ***c*** where ***r*** and ***c*** are single byte cursor address codes from the Wyse Cursor Address Table (Appendix A). Note that this command cannot position the cursor past column 95.‡

ESC **a** ***rr*** **R** ***ccc*** **C**
Moves the cursor to row ***rr*** and column ***ccc***. ***rr*** is the two digit decimal number of the row (from row 1 at the top). ***ccc*** is the three digit decimal number of the column (from column 1 at the left). Note that this command can address the entire 132 column screen.

‡ Only valid in Viewpoint Enhanced mode.

ESC _ c Moves the cursor to column ***c*** where ***c*** is a single byte column address from the Wyse Cursor Address Table (Appendix A). Note that this command cannot position the cursor past column 95. †

ESC [r Moves the cursor to row ***r*** where ***r*** is a single byte row address from the Wyse Cursor Address Table (Appendix A).

ESC] Selects page 0.

ESC } Selects page 1.

ESC – n r c
Selects page ***n*** (pages numbered from 0) and positions the cursor to row ***r***, column ***c***. ***r*** and ***c*** are single byte cursor address codes from the Wyse Cursor Address Table (Appendix A). To select pages higher than 9, use CHAR(***n*** +48).

ESC w n Selects page ***n***. Pages are numbered from 0 to 9.

ESC w C Page forward. If currently on the last page, the first page is selected.

ESC K Page forward. If currently on the last page, the first page is selected. †

ESC w B Page back. If currently on the first page, the last available page is selected.

ESC J Page back. If currently on the first page, the last available page is selected.

ESC w @ n r c
Selects page ***n*** (pages numbered from 0) and positions the cursor to row ***r***, column ***c***. ***r*** and ***c*** are single byte cursor address codes from the Wyse Cursor Address Table (Appendix A). To select pages higher than 9, use CHAR(***n*** +48).

† Only valid in Wyse 50 and Wyse 60 modes.

ESC* x A *r Splits screen horizontally at row ***r***, where ***r*** is a valid row code from the Wyse Cursor Address Table (Appendix A) or from the Viewpoint Cursor Address Table (Appendix B) if in Viewpoint Enhanced mode.

ESC* x 1 *r Splits screen horizontally at row ***r*** and clear screen, where ***r*** is a valid row code from the Wyse Cursor Address Table (Appendix A) or from the Viewpoint Cursor Address Table (Appendix B) if in Viewpoint Enhanced mode.

***ESC* x @** Redefine screen as one window.

***ESC* x 0** Redefine screen as one window and clear screen.

Erasing and Editing

SOH Clears the screen and returns the cursor to the home position (upper left corner of the screen).‡

FF Clears the screen and returns the cursor to the home position (upper left corner of the screen). Turns off the protected mode.‡

SUB Clears the screen and returns the cursor to the home position (upper left corner of the screen).†

***ESC* *** or ***ESC* +**
Clears the screen and move the cursor to the home position (upper left corner of the screen). Turns off the protect mode.

***ESC* T** or ***ESC* t**
Clears from the cursor position to the end of the current line.

***ESC* K** Clears from the cursor position to the end of the current line.‡

***ESC* Y** or ***ESC* y**
Clears from the cursor position to the end of the screen.†

‡ Only valid in Viewpoint Enhanced mode.
† Only valid in Wyse 50 and Wyse 60 modes.

ESC k Clears from the cursor position to the end of the screen.‡

ESC , Clears the screen to protected spaces. Turns off the protect mode. "No scroll" mode is turned off (auto scroll).

ESC E Insert line. A blank line is inserted before the current line. The current line and all lines below are moved down one row. The bottom line of the screen is lost. The cursor is moved to the left most column of the inserted line.

ESC Q Insert character. A blank is inserted at the current cursor position. All characters from the cursor position to the right are moved right one column.

ESC R Delete line. The current line is deleted. All lines below the current line are moved up one position. The bottom line is blank. The cursor is moved to the left most column.

ESC W Delete character. The character at the current cursor position is deleted. All characters to the right of the cursor are moved left one column and a blank is placed on the last column of the line.

ESC C N *w h*
Box rectangle. Current cursor location is upper left, ***w*** and ***h*** specify rectangle width and height.

ESC C G *r c*
Box rectangle. Current cursor location is one corner, ***r*** and ***c*** specify row and column of other corner.

ESC C F *r c x*
Clear unprotected rectangle. Cursor location is one corner, ***r*** and ***c*** specify row and column of other corner. Rectangle is cleared to character ***x***.

ESC C H *r c x*
Clear entire rectangle. Cursor location is one corner, ***r*** and ***c*** specify row and column of other corner. Rectangle is cleared to character ***x***.

ESC **C ^ *a sp* P *rr1* R *ccc1* C *rr2* R *ccc2* C *dp* P *rr3* R *ccc3* C**

Copy / swap / move rectangle. Action ***a*** is: 0 to swap, 1 to copy, 2 to move. Source page ***sp***, upper left corner ***rr1***, ***ccc1***; lower right corner ***rr2***, ***ccc2***; destination page ***dp***, upper left corner ***rr3***, ***ccc3***. All parameters are decimal numbers. Columns and rows are numbered from 1, pages are numbered from 0.

Video Attributes

Video attributes work differently in the Wyse 50 or Viewpoint Enhanced emulations than they do in the Wyse 60 emulation. Under Wyse 60, the attributes are non-embedded. In other words, they do not take up a character position on the screen. Under Wyse 50 or Viewpoint Enhanced, the attribute takes up a space and changes all following characters to the end of the screen or until another attribute character is encountered.

Under Viewpoint Enhanced mode, a single non-embedded (takes up no space) "tagged" attribute may be assigned and used. This "tagged" attribute is also the "protect" attribute.

ESC **0 *a*** Assigns the "tagged" video attribute. ***a*** is the video attribute code from the Viewpoint Attribute Code Table (Appendix B).‡

SO Start tagged attribute. All characters received after this code are displayed with the currently assigned tagged attribute. If the tagged attribute is changed, the attribute of the displayed characters also changes.‡

SI End tagged attribute. All characters received after this code are displayed with the normal video attribute.‡

‡ Only valid in Viewpoint Enhanced mode.

ESC A *n a* Sets the video attribute for any of the four application display areas. ***n*** is the display field code and ***a*** is the attribute code from the Wyse Attribute Code Table (Appendix A). The application display field codes are:

n	Display area
0	The main screen
1	The function key labeling line
2	The status line
3	The host message field

ESC G *a* Assign visual attribute. In Wyse 50 and ADDS VP mode, this command displays a space at the cursor position, then changes the visual attribute for all characters from this position until the end of the screen or until another attribute code is encountered. In Wyse 60 mode, this command selects the attribute for characters displayed after this command. The visual attribute is specified by the attribute code ***a*** from the Wyse Attribute Code Table (Appendix A).

ESC g *a*

Assign visual attribute. This command selects the attribute for characters displayed after this command. The visual attribute is specified by the attribute code ***a*** from the Wyse Attribute Code Table (Appendix A).‡

ESC G *n* or **ESC g *n*‡**

Assign line attribute. This is the same command as is used to assign visual attributes, but with different attribute codes. This sequence is used to specify character size for the current line. The line attribute code ***n*** is:

n	Attribute
@	normal size characters
A	double-wide characters
B	double-high characters, top
C	double-high characters, bottom
D	double-wide, double-high characters, top
E	double-wide, double-high characters, bottom

‡ Only valid in Viewpoint Enhanced mode.

ESC HT n Assign line attribute:‡

n	Attribute
0	normal size characters
1	double-wide characters
2	double-wide, double-high characters, top
3	double-wide, double-high characters, bottom

Protected Attributes

Protected attributes are used by some software to protect characters from being overwritten. If the terminal is running in its protected mode, the cursor cannot be positioned over protected characters and all unprotected characters can be cleared by one command.

ESC & Enable protect mode and set "no scroll" mode.

ESC ' Disable protect mode and set "auto scroll".

ESC) Start protect mode. All characters sent after this sequence become protected characters until the protect mode is turned off.

SO Start protect mode.‡

ESC (Stop protect mode. All characters sent after this sequence are unprotected characters.

SI Stop protect mode.‡

ESC ! *a* Writes all unprotected attributes with a specified attribute where ***a*** is a valid attribute code from the Wyse Attribute Code Table (Appendix A).

ESC . *c* Replaces all unprotected characters with the character ***c***.

‡ Only valid in Viewpoint Enhanced mode.

ESC **;** or ***ESC*** **:** or ***SUB***
Clears all unprotected characters.

ESC **,** Clears the screen to protected spaces. The protect mode is turned off and the auto scroll function is turned on.

ESC **V** Clears the entire cursor column to protected spaces.

ESC **cR** or ***ESC*** **cS**
Clears unprotected characters from the cursor position to the end of the current line. Attributes are not cleared.

ESC **cP** or ***ESC*** **cQ**
Clears unprotected characters from the cursor position to the end of the screen. Attributes are not cleared.

Line Graphics

ESC **H** ***g*** Display a line graphic character at the current cursor position. The graphic character is specified by graphic character code ***g*** from the Wyse Graphic Character Table (Appendix A). The line graphics characters can be used for drawing simple boxes on the screen in text mode. It should not be confused with the Tektronix graphic mode which is much more sophisticated and capable of drawing pie charts, scientific diagrams, etc.

ESC **H** ***STX*** Turn on line graphics mode. All characters received while the line graphics mode is on are interpreted as graphics characters according to the Wyse Graphic Character Table (Appendix A).

ESC **H** ***ETX*** Turn off line graphics mode.

Printer Control and Terminal Reports

ESC SPACE Reports the terminal identification to the host computer. Sends **50** followed by a ***CR***.

ESC ? Transmits the cursor address to the host computer. The cursor address is transmitted in 80 column format, followed by a ***CR***.

ESC / or ***ESC W*** `

Transmits the page number and cursor address to the host computer. The cursor address is transmitted in 80 column format, followed by a ***CR***.

ESC 4 Sends unprotected characters in the current row to the host computer, from the beginning of the row up to the cursor position. The row is terminated by a ***CR***.†

ESC 5 Sends all unprotected characters on the screen to the host computer, beginning with the upper-left corner, and ending at the cursor position. Each row except the last is terminated by a ***US*** control code. The last row is terminated by a ***CR***.†

ESC 6 Sends the current row to the host computer, from the beginning of the row, up to the cursor position. The row is terminated by a ***CR***.†

ESC 7 Sends the screen to the host computer, beginning with the upper-left corner, and ending at the cursor position. Each row except the last is terminated by a ***US*** control code. The last row is terminated by a ***CR***.

ESC L or ***ESC P*** or ***ESC p***

Prints the entire screen to the printer port.

ESC M Sends the character at the current cursor position to the host computer.

ESC S Sends a message unprotected. This function is not supported by AccuTerm.

† Not valid in Viewpoint Enhanced mode.

ESC* b** Sends the current cursor address to the host computer in 132 column format. No ***CR is sent after the address.

***ESC* s** Sends a message. This function is not supported by AccuTerm.

DC2 Auto print mode. Characters are displayed and printed. This command will not function if the printer selection is set to "No printer".

DC4 Cancel auto or transparent print mode. Note: This command will not turn the printer off if it was turned on by the Viewpoint Enhanced transparent print or ANSI print commands.

CAN Transparent print mode. Characters are printed, but not displayed.

***ESC* 3** Transparent print mode. Characters are printed, but not displayed.‡

***ESC* 4** Cancel transparent print mode.‡

***ESC* d #** Transparent print mode. Characters are printed, but not displayed.

***ESC* [? 5 i**

Transparent print mode (ANSI). Characters are printed, but not displayed. When AccuTerm receives this command, it goes into transparent print mode until it receives the ANSI print off sequence below. This command is useful for printing graphics data to the printer. Since AccuTerm requires the specific ANSI cancel command below to exit transparent print mode, there is less chance of a control character interrupting the graphic printing.

***ESC* [? 4 i**

Cancel transparent print mode (ANSI).

‡ Only valid in Viewpoint Enhanced mode.

Programming Function Keys

The Wyse 50, Wyse 60 and Viewpoint Enhanced emulations support the ability to download function key values from the host computer. The function key programming consists of two steps, downloading the actual values which the key will send to the host whenever it is pressed or downloading a descriptive function key label that is displayed on the function key labeling line.

When programming the function keys or function key labels, all characters (including control characters) may be included in the sequence, except for the terminator (***DEL*** for keys, ***CR*** for labels).

To clear a programmed function key or label, send the same command used for programming the key or label, but omit the sequence.

ESC z k sequence DEL

Program function key ***k*** to send ***sequence*** to host when pressed. The function key codes are specified in the Wyse Function Key Table (Appendix A).

ESC z f label CR

Program function key label field ***f*** as ***label***. The field codes are specified in the Wyse Function Key Table (Appendix A).

ESC z 0 k sequence DEL

Program function or keypad key ***k*** to send ***sequence*** to host when pressed. The function and keypad key codes are specified in the Wyse Function Key Table (Appendix A).

ESC z (text CR

Sets the unshifted label line to ***text***. If ***text*** is omitted, clears the unshifted label line.

ESC z) text CR

Sets the shifted label line to ***text***. If ***text*** is omitted, clears the shifted label line.

ESC _ f label EM

Program function key label field ***f*** as ***label***. The function key field codes are specified in the Viewpoint Function Key Table (Appendix B).‡

ESC f text CR

Sets the unshifted label line to ***text***. If ***text*** is omitted, clears the unshifted label line.

ESC % k 0n sequence EM

Program function or keypad key ***k*** to send ***sequence*** to host when pressed. The function and keypad key codes are specified in the Viewpoint Function Key Table (Appendix B).‡

ESC c U Clear all redefinable key definitions to their default values.

ESC Z ~ k Read programmable key definition for key ***k***.

‡ Only valid in Viewpoint Enhanced mode.

ADDS PROGRAMMING

This section describes the command sequences for programming the ADDS Viewpoint A2, Viewpoint 60 and Procomm VP60 terminal emulations. These emulations use a common command set with a few differences. The differences are noted.

Operating Modes

ESC **B**	Status line off.†
ESC **b**	Status line on.†
ESC **D**	Screen display off.†
ESC **d**	Screen display on.†
ETB	Turns off the cursor.
CAN	Turns on the cursor.
GS	If the graphics mode is enabled, this command puts the terminal in the Tektronix emulation mode. If the graphics mode is disabled, this command is ignored.
ESC **% ! 0**	Enters Tektronix 4100 graphics mode.
CAN	Exits from the Tektronix graphics emulation mode.

† Not valid in ADDS Viewpoint A2 mode.

Cursor Positioning

ACK Cursor right. If the cursor is on the last column of the line, the cursor will wrap to the next line, possibly scrolling the screen up.

BS or ***NAK*** Cursor left. If the cursor is at the beginning of the line, it is moved up to the last column of the previous line. If the cursor is at the home position, it is moved to the lower right hand corner of the screen.

LF Cursor down. If the cursor is on the last line of the screen and the "no scroll" mode is turned off, the screen will scroll up one line. Otherwise, the cursor will move to the top line of the screen.

SUB Cursor up. If the cursor is at the top row, it is moved to the bottom row.

CR Moves the cursor to the first column (column zero) of the current row.

VT r Moves cursor to row ***r***, where ***r*** is a valid row code from the Viewpoint Cursor Address Table (Appendix B).

DLE c Moves the cursor to column ***c*** where ***c*** is a valid column in the from the Viewpoint Cursor Address Table (Appendix B).

ESC Y r c Moves the cursor to row ***r*** and column ***c*** where ***r*** and ***c*** are single byte cursor address codes from the Wyse Cursor Address Table (Appendix A). Note that this command cannot position the cursor past column 95.

Erasing and Editing

FF	Clears the screen and returns the cursor to the home position (upper left corner of the screen). Turns off the protected mode.
ESC **K**	Clears from the cursor position to the end of the current line.
ESC **k**	Clears from the cursor position to the end of the screen.
ESC **M**	Insert line. A blank line is inserted before the current line. The current line and all lines below are moved down one row. The bottom line of the screen is lost. The cursor is moved to the left most column of the inserted line.†
ESC **F**	Insert character. A blank is inserted at the current cursor position. All characters from the cursor position to the right are moved right one column.†
ESC **l**	Delete line. The current line is deleted. All lines below the current line are moved up one position. The bottom line is blank. The cursor is moved to the left most column.†
ESC **E**	Delete character. The character at the current cursor position is deleted. All characters to the right of the cursor are moved left one column and a blank is placed on the last column of the line.†

† Not valid in ADDS Viewpoint A2 mode.

Video Attributes

The Viewpoint A2, Viewpoint 60 and Procomm VP60 have different ways of programming video attributes.

The ADDS Viewpoint A2 terminal supports video attributes through the use of "tagged" characters. When a character is received from the host, it is either tagged or normal. Tagged characters are displayed with the currently assigned tagged video attribute.

Programming video attributes involves two separate steps. First, the tag attribute must be assigned. Next, in order to display characters in the assigned tag attribute, a "start tag attribute" code must be sent. To send characters in the normal attribute, an "end tag attribute" code must be sent. These two steps are independent and can be executed in either order.

Any time the tagged attribute is changed, all of the tagged characters already on the screen will be displayed in the new attribute. However, only one video attribute can be displayed at a time. This limitation only exists in the standard Viewpoint emulation mode. In the Viewpoint enhanced mode, both the Viewpoint and Wyse attributes can be used at the same time.

The ADDS Viewpoint 60 (and Procomm VP60) emulations use the same escape sequence as the Viewpoint A2 but it is interpreted differently. Under ADDS Viewpoint 60 mode, the attribute takes up a space and changes all following characters to the end of the screen or until another attribute character is encountered. It works similar to the Wyse 50 video attributes. Under Procomm VP60, multiple visual attributes may be displayed at the same time, but the attribute character does not use a space on the screen, and does not affect any previously displayed characters. It works similar to the Wyse 60 video attributes.

SO Start tagged attribute. All characters received after this code are displayed with the currently assigned tagged attribute. If the tagged attribute is changed, the attribute of the displayed characters also changes.

SI End tagged attribute. All characters received after this code are displayed with the normal video attribute.

ESC 0 *a* **ADDS Viewpoint A2:** Assign visual attribute. This command sets the current visual attribute. All characters which are "tagged" are displayed on this attribute. The attribute is specified by the attribute code ***a*** from the Viewpoint Attribute Code Table (Appendix B).

ADDS Viewpoint 60: Assign visual attribute. This command is used to start a specific attribute. This command changes all characters from the current position until the end of the screen or until another attribute code character is encountered. The attribute uses a screen position. The attribute is specified by the attribute code ***a*** from the Viewpoint Attribute Code Table (Appendix B).

Procomm VP60: Assign visual attribute. This command is used to start a specific attribute. The attribute does not use a screen position. All characters output after this command are displayed in the specified attribute. The attribute code is specified by the attribute code ***a*** from the Viewpoint Attribute Code Table (Appendix B).

Line Graphics

ESC 1 Turn on line graphics mode. All characters received while the line graphics mode is on are interpreted as graphics characters according to the Viewpoint 60 Graphic Character Table (Appendix B).†

ESC 2 Turn off line graphics mode.†

† Not valid in ADDS Viewpoint A2 mode.

Printer Control

DC2 Auto print mode. Characters are displayed and printed. This command will not function if the printer option on the setup menu is set to "None".

DC4 Cancel auto or transparent print mode. Note: This command will not turn the printer off if it was turned on by the Viewpoint Enhanced transparent print or ANSI print commands.

ESC **3** Transparent print mode. Characters are printed, but not displayed.

ESC **4** Cancel transparent print mode.

ESC **[? 5 i** Transparent print mode (ANSI). Characters are printed, but not displayed. When AccuTerm receives this command, it goes into transparent print mode until it receives the ANSI print off sequence below. This command is useful for printing graphics data to the printer. Since AccuTerm requires the specific ANSI cancel command below to exit transparent print mode, there is less chance of a control character interrupting the graphic printing.

ESC **[? 4 i** Cancel transparent print mode (ANSI).

ANSI PROGRAMMING

The AccuTerm ANSI emulations provide DEC VT52, VT100, VT220, VT320, VT420, SCO Console, Linux Console and ANSI BBS emulations. The VT220, VT320 and VT420 emulations support international characters, 7 and 8 bit control codes, programmable function keys and a robust command set. VT100 supports 7 bit control codes and a subset of the VT220 command set.

The SCO Console emulation uses the PC (OEM) character set and supports programmable function keys and text and background colors.

The Linux Console emulation uses the ANSI character set, and includes support for programmable function keys, text and background colors, and mouse.

The ANSI BBS emulation supports the PC (OEM) character set and the attribute and cursor commands supported by the DOS ANSI.SYS device driver (AccuTerm does not use the ANSI.SYS driver). The ANSI BBS emulation only supports 7 bit escape sequences.

When AccuTerm is emulating a VT220, VT320 or VT420 terminal or SCO or Linux Console, it will respond to 7-bit and 8-bit control codes (hex 00-1F and 80-9F). For convenience, command sequences which may use 8-bit control codes are documented using the 8-bit control code. For every 8-bit control code, an equivalent 7-bit escape sequence may also be used, as shown in the following table:

CSI	=	***ESC*** **[**
SS3	=	***ESC*** **O**
DCS	=	***ESC*** **P**
ST	=	***ESC*** ****

This chapter documents commands used by all of the ANSI emulations supported by AccuTerm. Not every command is valid for all ANSI emulations. Those commands which are only valid for certain emulations are shown with a reference note after the command definition. These reference are indicated by a superscript, and are VT100[1], VT220[2], VT320[3], VT420[4], ANSI BBS[A], Linux Console[L], and SCO Console[5].

The AccuTerm ANSI emulations operate in a variety of modes. Some of the default operating modes are determined by settings in the AccuTerm configuration. Most of these modes can be changed by commands received from the host. Operating modes which can be selected in the Settings dialog and saved in a configuration file are:

- VT52, VT100, VT220, VT320, VT420, SCO Console, Linux Console or ANSI BBS emulation
- 80 (normal) or 132 (extended) columns
- Automatic wrap at end of line
- Send 7 or 8 bit control codes
- Numeric keypad sends “application” codes instead of numbers
- Cursor keys send “application” codes instead of cursor codes
- Backspace key sends DEL control code instead of BS control code

The following operating modes are not determined by settings in the AccuTerm configuration, but may be changed by sending the appropriate ANSI command sequence:

- Keyboard: default is unlocked.
- Insert/Replace: default is replace.
- Line feed / New line: default is line feed.
- Origin: default is absolute.
- Tabs: default is every 8 columns.
- Cursor: default is on.
- Print extent: default is full screen.
- Printer form-feed: default is off.

- Graphics mode: default is off.

AccuTerm VT220, VT320 and VT420 emulations support 5 character sets: ASCII, International, ISO-Latin1, Graphics and Scientific. The default character set assignment is G0=ASCII, G1=ASCII, G2=International and G3=ASCII. The GL set (hex 20-7E) defaults to G0 (ASCII) and the GR set (hex A0-FE) defaults to G2 (International).

Operating Modes

ESC **c** Hard reset. Re-reads AccuTerm configuration file, then resets all operating modes and character sets to their default values. Clears the screen and I/O buffer.

CSI **! p** Soft reset. Resets all operating modes and character sets to their default values.[234]

CSI ***n*** **+ p** Secure reset. Re-reads AccuTerm configuration file, then resets all operating modes and character sets to their default values. Clears the screen and I/O buffer. If ***n*** is non-zero, then AccuTerm responds by sending ***CSI*** ***n*** *** q** back to the host. The value of ***n*** must be between 0 and 16383.[1234]

ESC SPACE **F**
Causes AccuTerm to send 7-bit control codes: [234]

CSI	=	***ESC*** **[**
SS3	=	***ESC*** **O**
DCS	=	***ESC*** **P**
ST	=	***ESC*** ****

ESC SPACE **G**
Causes AccuTerm to send 8-bit control codes ***CSI*** , ***SS3***, ***DCS*** and ***ST***. [234]

CSI **6 1 " p**
Changes the emulation to VT100.[234]

CSI **6 2 ;** ***n*** **" p**
Changes the emulation to VT220. If ***n*** = 1, AccuTerm will

equivalent send 7-bit escape sequences for control codes ***CSI*** , ***SS3***, ***DCS*** and ***ST***; otherwise, 8-bit control codes will be sent.[234]

CSI **6 3 ; *n* " p**
Changes the emulation to VT320. If ***n*** = 1, AccuTerm will equivalent send 7-bit escape sequences for control codes ***CSI*** , ***SS3***, ***DCS*** and ***ST***; otherwise, 8-bit control codes will be sent.[234]

CSI **6 4 ; *n* " p**
Changes the emulation to VT420. If ***n*** = 1, AccuTerm will equivalent send 7-bit escape sequences for control codes ***CSI*** , ***SS3***, ***DCS*** and ***ST***; otherwise, 8-bit control codes will be sent.[234]

CSI **2 h** Locks the keyboard.

CSI **2 l** Unlocks the keyboard.

CSI **3 h** Enable display of control characters as symbols.

CSI **3 l** Disable display of control characters (execute control characters.

CSI **4 h** Insert mode on.

CSI **4 l** Insert mode off.

CSI **12 h** Full duplex (no local echo).

CSI **12 l** Half duplex (local echo).

CSI **20 h** Process ***LF***, ***VT*** and ***FF*** as "new line"; that is perform carriage return and line feed.

CSI **20 l** Process ***LF***, ***VT*** and ***FF*** as line feed.

CSI **42 h** Changes emulation to Wyse 60.

CSI **? 1 h** Cursor keys return application codes.

CSI **? 1 l** Cursor keys return cursor codes.

CSI **? 2 l** Enter VT52 emulation mode.

CSI ? 3 h Extended video mode (132 columns).

CSI ? 3 l Normal video mode (80 columns).

CSI ? 5 h Light background, dark text.

CSI ? 5 l Dark background, light text.

CSI ? 6 h Causes cursor positioning to be relative to the currently defined scrolling region.

CSI ? 6 l Causes cursor positioning to be absolute (not relative).

CSI ? 7 h Sets autowrap mode. When the cursor is on the last character of a line, receipt of another character causes the cursor to move to the first column of the next line.

CSI ? 7 l Resets autowrap mode. The cursor will not move past the last column of the line upon receipt of another character.

CSI ? 9 h or *CSI* 0 $ ~
Status line off. [234]

CSI ? 9 l Status line on.[234]

CSI 1 $ ~ Display local status line. [234]

CSI 2 $ ~ Display host-writable status line. [234]

CSI 0 $ } Data sent to screen's data area. [234]

CSI 1 $ } Data sent to host-writable status line. [234]

CSI ? 18 h
Causes a form-feed character (hex 0C) to be sent to the printer after each print screen.

CSI ? 18 l
No character is sent to the printer after each print screen.

CSI ? 19 h
Print screen command causes the full screen to be printed.

CSI **? 19 l**
Print screen command only prints the currently defined scrolling region.

CSI **? 25 h**
Cursor on.

CSI **? 25 l**
Cursor off.

CSI **? 38 h** or ***GS*** or ***ESC*** **1**
Enters Tektronix 4014 graphics mode.

CSI **? 38 l** or ***CAN*** or ***ESC*** **2**
Exits Tektronix 4014 graphics mode.

CSI **? 66 h**
Cursor keys return application codes.[2,3,4]

CSI **? 66 l**
Cursor keys return cursor codes.[2,3,4]

CSI **? 67 h**
Backspace keys sends ***BS*** control code.[3,4]

CSI **? 67 l**
Backspace keys sends ***DEL*** control code.[3,4]

CSI **? 69 h**
Enables vertical split screen mode.[4]

CSI **? 69 l**
Disables vertical split screen mode.[4]

CSI **? 95 h**
Do not clear screen when column mode changes.[3,4]

CSI **? 95 l**
Clear screen when column mode changes.[3,4]

CSI **? 109 h**
Set **CapsLock** keyboard state.

CSI **? 109 l**
Clear **CapsLock** keyboard state.

***ESC* % ! 0**
Enters Tektronix 4100 graphics mode.

***CSI* 0 *SP* q** or ***CSI* 1 *SP* q** or ***CSI* 2 *SP* q**
Select block cursor. [234]

***CSI* 3 *SP* q** or ***CSI* 4 *SP* q**
Select underscore cursor. [234]

***CSI n* $ |**
Set number of columns to ***n***. [34]

***CSI n* t** or ***CSI n* * |**
Set number of rows to ***n***. [34]

***CSI n* , q**
Sets the terminal ID returned in response to the DA1 command. Valid values for ***n*** are: 0 = VT100, 1 = VT101, 2 = VT102, 5 = VT220 and 6 = VT320. [34]

***CSI n* * x**
Selects if visual attributes which are modified by the DECCARA or DECRARA commands are contained within the rectangular area defined by the beginning and ending positions. If ***n*** = 0 or 1 then beginning and ending positions indicate locations in the character stream displayed on the screen; if ***n*** = 2, then the beginning and ending positions indicate the upper-left and lower-right corners of a rectanle. [L]

***CSI* ? 9 h** Enable mouse reporting. [L]

***CSI* ? 9 l** Disable mouse reporting. [L]

***CSI n* = L**
Set erase mode. If ***n*** = 0, new lines and clear screen are filled with the current background color. Otherwise, new lines and clear screen fill with the screen background color. [S]

***ESC* =** Numeric pad returns application codes.

***ESC* >** Numeric pad returns numbers.

Character Set Selection

ESC (B Sets character set G0 to ASCII (default).[1234L]

ESC) B Sets character set G1 to ASCII (default).[1234L]

ESC * B Sets character set G2 to ASCII.[234L]

ESC + B Sets character set G3 to ASCII.[234L]

ESC (< Sets character set G0 to Multinational.[234L]

ESC) < Sets character set G1 to Multinational.[234L]

ESC * < Sets character set G2 to Multinational (default).[234]

ESC + < Sets character set G3 to Multinational.[234]

ESC (A Sets character set G0 to ISO-Latin1.[1L]

ESC) A Sets character set G1 to ISO-Latin1.[1L]

ESC - A Sets character set G1 to ISO-Latin1.[234]

ESC . A Sets character set G2 to ISO-Latin1.[234]

ESC / A Sets character set G3 to ISO-Latin1.[234]

ESC (0 Sets character set G0 to graphics.[1234L]

ESC) 0 Sets character set G1 to graphics.[1234L]

ESC * 0 Sets character set G2 to graphics.[234]

ESC + 0 Sets character set G3 to graphics.[234]

ESC (S Sets character set G0 to scientific.[234]

ESC) S Sets character set G1 to scientific.[234]

***ESC* * S** Sets character set G2 to scientific.[234]

***ESC* + S** Sets character set G3 to scientific.[234]

***ESC* (% 5** Sets character set G0 to Multinational.[1234]

***ESC*) % 5** Sets character set G1 to Multinational.[1234]

***ESC* * % 5** Sets character set G2 to Multinational.[234]

***ESC* + % 5** Sets character set G3 to Multinational.[234]

***ESC* (K** or ***ESC* (U**
Sets character set G0 to codepage 437.[L]

***ESC*) K** or ***ESC*) U**
Sets character set G1 to codepage 437.[L]

ESC* (*VT Sets character set G0 to codepage 437.[LS]

ESC*)*VT Sets character set G1 to codepage 437.[LS]

ESC* (*FF Sets character set G0 to upper half of codepage 437.[LS]

ESC*)*FF Sets character set G1 to upper half of codepage 437.[LS]

SI Invoke character set G0 into GL (the GL set corresponds to character codes in the range of hex 20-7E); this is the default.[1234LS]

SO Invoke character set G1 into GL.[1234LS]

***ESC* ~** Invokes character set G1 into GR (the GR set corresponds to character codes in the range of hex A0-FE).[234]

***ESC* n** Invokes character set G2 into GL.[234]

***ESC* }** Invokes character set G2 into GR (default).[234]

***ESC* o** Invokes character set G3 into GL.[234]

***ESC* |** Invokes character set G3 into GR.[234]

SS2 or ***ESC*** **N**

Invokes character set G2 into GL for the next character only.[234]

SS3 or ***ESC*** **O**

Invokes character set G3 into GL for the next character only.[234]

Cursor Positioning

BS Moves the cursor back one space. If the cursor is at the beginning of the line, no action occurs.

HT Moves the cursor to the next programmed tab stop. If there are no more tab stops, the cursor moves to the right margin.

LF Moves the cursor down one line. If the cursor is on the last line of the scrolling region, the screen will scroll up one line.

VT Same as ***LF***.

FF Same as ***LF***.

CR Moves the cursor to the left margin of the current line.

NEL or ***ESC*** **E**
Moves the cursor to the first column of the next line of the scrolling region. If the cursor is on the last line of the scrolling region, the screen will scroll up one line.

IND or ***ESC*** **D**
Moves cursor down one line in the same column. If the cursor is on the last line of the scrolling region, the screen will scroll up one line.

CSI n **T** Moves cursor down ***n*** lines in the same column. If the cursor is moved to the last line of the scrolling region, the screen will scroll up.[s]

RI or ***ESC*** **M**
Moves the cursor up one line in the same column. If the cursor is on the first line of the scrolling region, the screen will scroll down one line.

CSI n **S** Moves the cursor up ***n*** lines in the same column. If the cursor is moved to the first line of the scrolling region, the screen will scroll down.[s]

***CSI n* B** or ***CSI n* e**
Moves cursor down ***n*** lines in the same column.

CSI n* E** Moves cursor down ***n lines and to the first column.[1234LS]

CSI n* A** Moves the cursor up ***n lines in the same column.

CSI n* F** Moves the cursor up ***n lines and to the first column.[1234LS]

CSI n* D** Moves cursor left ***n columns.

***CSI n* C** or ***CSI n* a**
Moves cursor right ***n*** columns.

***CSI line* ; *column* H** or ***CSI line* ; *column* f**
Move cursor to line ***line***, column ***column***.

CSI n* d** Moves cursor to line ***n.[1234LS]

***CSI n* G** or ***CSI n* `**
Moves cursor to column ***n***.[1234LS]

HTS or ***ESC* H** or ***CSI* 0 W**
Sets a tab stop at the column where the cursor is.

***CSI* 0 g** or ***CSI* 2 W**
Clears a tab stop at the column where the cursor is.[1234LS]

***CSI* 3 g** or ***CSI* 5 W**
Clears all tab stops.[1234LS]

***CSI* ? 5 W** Sets tab stops every 8th column.[234]

CSI n* I** Move forward ***n tab stops.[234]

CSI n* Z** Move backward ***n tab stops.[234]

***CSI n* V** or ***CSI n SPACE* R**
Display a preceding page. If ***n*** is 0 or 1, the previous page is displayed, otherwise ***n*** specifies the number of pages back to be displayed.[234]

CSI n **U** or ***CSI n SPACE*** **Q**

Displays a following page. If ***n*** is 0 or 1, the next page is displayed, otherwise ***n*** specifies the number of pages forward to be displayed.[234]

CSI n SPACE **P**

Display page ***n***.[234]

CSI top **;** ***bottom*** **r**

Set scrolling region. The first line of the scrolling region is set to ***top***; the last line to ***bottom***. Default values for ***top*** and ***bottom*** are 1 and 24 respectively. Once the scrolling region is defined, if origin mode is set to relative, the cursor may be positioned into, but not out of, the scrolling region.[1234LS]

ESC **7** or ***CSI*** **s**[ALS]

Save state (cursor position, video attribute, character set, autowrap, origin mode and protect mode).

ESC **8** or ***CSI*** **u**[ALS]

Restore state.

CSI left **;** ***right*** **s**

Sets the left and right margins to define a *horizontal* scrolling region. This command only works when vertical split screen mode is enabled.

ESC **6** Move cursor left one column. If the cursor is at the left margin, all data within the margin scrolls right one column. The column that shifted past the right margin is lost.[4]

ESC **9** Move cursor right one column. If the cursor is at the right margin, all data within the margin scrolls left one column. The column that shifted past the left margin is lost.[4]

ESC **8** Fill screen with upper-case “E”.

Erasing and Editing

CSI **0 J** Erases from the cursor to the end of the screen, including the cursor position.

CSI **1 J** Erases from the beginning of the screen to the cursor, including the cursor position.

CSI **2 J** Erases the entire screen (the cursor position is not moved).

CSI **0 K** Erases from the cursor to the end of the line, including the cursor position.

CSI **1 K** Erases from the beginning of the line to the cursor, including the cursor position.

CSI **2 K** Erases the entire line (the cursor is not moved).

CSI **? 0 J** Erases all unprotected characters from the cursor to the end of the screen, including the cursor position.

CSI **? 1 J** Erases all unprotected characters from the beginning of the screen to the cursor, including the cursor position.

CSI **? 2 J** Erases all unprotected characters on the entire screen (the cursor position is not moved).

CSI **? 0 K** Erases all unprotected characters from the cursor to the end of the line, including the cursor position.

CSI **? 1 K** Erases all unprotected characters from the beginning of the line to the cursor, including the cursor position.

CSI **? 2 K** Erases all unprotected characters on the line (the cursor is not moved).

CSI ***n*** **X** Erases ***n*** characters, beginning with the current cursor position.[234LS]

CSI ***n*** **@** Insert ***n*** blank characters beginning at the current cursor position.[234LS]

CSI *n* L Insert ***n*** blank lines beginning at the cursor line.[1234LS]

CSI *n* P Delete ***n*** characters beginning at the current cursor position.[1234LS]

CSI *n* M Delete ***n*** lines beginning at the cursor line.[1234LS]

CSI *n* ' } Insert ***n*** columns into the scrolling region beginning at the column that has the cursor.[4]

CSI *n* ' ~ Delete ***n*** columns from the scrolling region beginning at the column that has the cursor.[4]

CSI *top* ; *left* ; *bottom* ; *right* ; *attr1* ; ... ; *attrn* $ r
Changes visual attributes in an area defined by ***top***, ***left***, ***bottom***, ***right***. The area to be changed is either a character stream or rectangle as defined by the DECSACE command. See Appendix C for a table of the attribute codes.[4]

CSI *top* ; *left* ; *bottom* ; *right* ; *attr1* ; ... ; *attrn* $ t
Reverse visual attributes in an area defined by ***top***, ***left***, ***bottom***, ***right***. The area to be changed is either a character stream or rectangle as defined by the DECSACE command. See Appendix C for a table of the attribute codes.[4]

CSI *top* ; *left* ; *bottom* ; *right* $ z
Erase characters in rectangle defined by ***top***, ***left***, ***bottom***, ***right***.[4]

CSI *top* ; *left* ; *bottom* ; *right* $ {
Erase unprotected characters in rectangle defined by ***top***, ***left***, ***bottom***, ***right***.[4]

CSI *ch* ; *top* ; *left* ; *bottom* ; *right* $ x
Fill rectangle defined by ***top***, ***left***, ***bottom***, ***right*** with character ***ch*** (ASCII code).[4]

CSI *stop* ; *sleft* ; *sbottom* ; *sright* ; *spg* ; *dtop* ; *dleft* ; *dpg* $ v
Copy rectangle defined by ***stop***, ***sleft***, ***sbottom***, ***sright***, from page ***spg*** to page ***dpg*** at location ***dtop***, ***dleft***.[4]

Video Attributes

CSI ***n*** **;** ***n*** **. . . m**
Selects video attributes and/or character foreground and background colors according to the ANSI Attribute Code Table (Appendix C). Characters received after this command are displayed in the selected video attribute. If multiple parameters are used, their effects are cumulative (e.g. 0;4;5 selects blinking-underlined). A parameter value of 0 resets all attributes.

CSI 0 " q Unprotected mode. Characters received after this command are erasable using the "erase unprotected characters" command.[234]

CSI 1 " q Protected mode. Characters received after this command are not erasable using the "erase unprotected characters" command.[234]

ESC # 3 Double-high line top. Causes the line containing the cursor to display the top half of a double-high line.[1234]

ESC # 4 Double-high line bottom. Causes the line containing the cursor to display the bottom half of a double-high line.[1234]

ESC # 5 Single-width line. Causes the line containing the cursor to display normal width characters.[1234]

ESC # 6 Double-width line. Causes the line containing the cursor to display double width characters.[1234]

CSI = *n* F Sets the current normal foreground color to ***n***. Refer to "AccuTerm Programming" chapter for color values.[5]

CSI = *n* G Sets the current normal background color to ***n***.[5]

CSI = *n* H Sets the current reverse foreground color to ***n***.[5]

CSI = *n* I Sets the current reverse background color to ***n***.[5]

ESC **] P** ***nrrggbb***

Set Linux Console palette. ***N*** is a single hex digit indicating which palette entry to set; ***rrggbb*** are the hex RGB color values to be set. [L]

ESC **] R**

Reset Linux Console palette. [L]

Printer Control

CSI **0 i** Prints the screen display. Either the full screen or scrolling region may be selected, and a form feed may be sent after printing (see Operating Modes section.

CSI **10 i** Prints the screen display ignoring the print extent in effect.[34]

CSI **? 1 i** Prints the current cursor line.

CSI **5 i** Transparent print mode. Characters are printed, but not displayed.

CSI **? 5 i** Auto print mode. When the "Auto-print works just like a VT terminal" setting is enabled, a line is printed from the screen when the cursor moves off that line with an ***LF***, ***FF***, or ***VT*** control code, or an autowrap occurs. Otherwise auto-print mode works just like Wyse auto-print mode.

CSI **4 i** or ***CSI*** **? 4 i**

Cancel transparent or auto print mode.

CSI **2 i** Send screen to host.[234]

Terminal Reports

CSI **0 c** or ***ESC*** **Z**
Request primary device attributes. Depending on the current emulation and terminal ID in effect, AccuTerm will respond:[1234L]
VT100: ***CSI*** **? 1 ; 2 c**
VT220: ***CSI*** **? 6 2 ; 1 ; 2 ; 6 ; 8 c**
VT320: ***CSI*** **? 6 3 ; 1 ; 2 ; 6 ; 8 c**
VT420: ***CSI*** **? 6 4 ; 1 ; 2 ; 6 ; 8 c**
Linux: ***CSI*** **? 6 c**

CSI **> 0 c** Request secondary device attributes. AccuTerm will respond:[234]
CSI **> 41 ; 4 ; 1 c**

CSI **= 0 c** Request tertiary device attributes. AccuTerm will respond (***XXXXXXXX*** is the product serial number in hexadecimal):[234]
DCS **! |1** ***XXXXXXXX*** ***ST***

CSI **5 n** Request terminal status. AccuTerm will respond:
CSI **0 n**

CSI **6 n** Request cursor position. AccuTerm will respond:
CSI ***line*** **;** ***column*** **R**

CSI **? 6 n** Request cursor position and page. AccuTerm will respond:
CSI ***line*** **;** ***column*** **;** ***page*** **R**

CSI **? 1 5 n**
Request printer status. If a printer is defined for the current configuration, AccuTerm will respond:[234]
CSI **? 1 0 n**
If no printer is defined the response will be:
CSI **? 1 3 n**

CSI **? 2 5 n**
Request status of user-defined keys. AccuTerm will respond:[234]
CSI **? 2 0 n**

CSI ? 2 6 n

Request keyboard status. AccuTerm will respond:[2][3][4]

CSI ? 2 7 ; 0 ; 0 ; 0 n

CSI ? 5 5 n

Request locator status. If the mouse is currently enabled, AccuTerm will respond:[2][3][4]

CSI ? 5 0 n

If the mouse is not enabled, the response will be:

CSI ? 5 3 n

CSI ? 5 6 n

Request locator device type. If the mouse is currently enabled, AccuTerm will respond:[2][3][4]

CSI ? 5 7 ; 1 n

If the mouse is not enabled, the response will be:

CSI ? 5 7 ; 0 n

CSI ? 6 2 n

Request macro space. AccuTerm will respond:[2][3][4]

CSI 0 * {

CSI ? 6 3 ; *id* n

Request memory checksum. AccuTerm will respond:[2][3][4]

DCS id ! ~ 0 0 0 0 *ST*

CSI ? 7 5 n

Request data integrity. AccuTerm will respond:[2][3][4]

CSI ? 7 0 n

CSI " v

Request displayed extent. AccuTerm will respond:[2][4]

CSI rows ; *columns* ; 1 ; 1 ; *page* " w

CSI n $ p

Request state of ANSI mode ***n***. AccuTerm will respond:[3][4]

CSI n ; *value* $ y

CSI ? *n* $ p

Request state of private mode ***n***. AccuTerm will respond:[3][4]

CSI ? *n* ; *value* $ y

CSI **+ x**

Request function key free memory. AccuTerm will respond:[34]
CSI **804 ; 804 + y**

CSI **1 $ w**

Request cursor information report. AccuTerm will respond with a string encoded with the current cursor state including position, page, attribute and character set. The string returned can be use to restore the cursor state. The response string is:[34]
DCS **1 $ u** ***string*** ***ST***

DCS **1 $ t** ***string*** ***ST***

Restore cursor state. Use ***string*** returned from previous command.[34]

CSI **2 $ w**

Request tabstop report. AccuTerm will respond with a string encoded with the current tab settings. The string returned can be use to restore the tabs. The response string is:[34]
DCS **2 $ u** ***string*** ***ST***

DCS **2 $ t** ***string*** ***ST***

Restore tabstops. Use ***string*** returned from previous command.[34]

DCS **$ q** ***setting*** ***ST***

Request setting. ***Setting*** is formed by taking the last one or two non-numeric characters of an ANSI command. The response is:[34]
DCS **0 $ r** ***string*** ***ST***
You can use the response ***string*** to restore the setting; Simply add ***CSI*** to the beginning of the string and send it back to the terminal.

CSI **1 $ u**

Request terminal state. The complete terminal state (cursor position, character set, attributes, screen size, key lock, etc.) is encoded into a string and returned to the host. You can use this string to restore the state later. The response is:[34]
DCS **1 $ s** ***string*** ***ST***

DCS **1 $ p** ***string*** ***ST***

Restore terminal state. ***String*** is the value returned from the previous command.[34]

Programming Function Keys

The AccuTerm VT220, VT320, VT420, Linux Console and SCO Console emulations support the ability to download up to 15 function key values (the shifted function keys **F6-F20)** from the host computer. The following device control sequence is used to program the function keys.

DCS n | key / value ; . . . key / value ST

If ***n*** is 1, old key definitions are replaced by new definitions; if ***n*** is 0, the definition of all 15 shifted function keys are cleared before loading any new definitions. The function key to be programmed is specified by ***key*** according to the ANSI Function Key Table (Appendix C), and the sequence for that key is specified by ***value***. ***Value*** is specified in hexadecimal.

VT320 and VT420 emulations provide an extended method to program function and editing keys:

DCS **" x** ***key / mod /*** **100** ***/ value /*** **0 ;** ***. . . ST***

The key to be programmed is specified by ***key*** according to the ANSI Extended Key Table (Appendix C), and any modifier keys (**SHIFT**, **CTRL**, etc.) are specificed by ***mod***, according to the same table. The sequence for the key is specified by ***value*** in hexadecimal.

In addition, the VT420 emulation has user-definable macros. Use the following command to define macros.

DCS id ; dt ; fn ; en ! z value ST

id is a macro identifier, which must be in the range of 0 to 63. If ***dt*** is 1, then all current macros are deleted before the new macro is defined. ***en*** specifies the encoding: 0 for ASCII text, 1 for hexidecimal. ***Value*** is the macro contents in the specified encoding format.

Invoking a macro has the same effect as if the terminal had received the macro contents from the host. The following command may be use to invoke a defined macro:

CSI id * z[4]

The SCO Console emulation uses the following sequence to program function keys:

ESC Q key delim value delim

key is a single ASCII character which designates which key to program. Function keys **F1** to **F12** are selected by specifying **0** to **;** . For **SHIFT**+**F1** to **SHIFT**+**F12** use **<** to **G**. For **CTRL**+**F1** to **CTRL**+**F12** use **H** to **S**. For **CTRL**+**SHIFT**+**F1** to **CTRL**+**SHIFT**+**F12** use **T** to **_**. ***Delim*** is a single character delimiter which encloses ***value***.

CSI 2 + z Restores default values to programmed keys.[3]

To program the Answerback message, use the following device control sequence:

DCS 1 v value ST

Note: the "Map F11 through F20 …" check box in the **Keyboard Settings** category, **Settings** dialog, must be checked to access programmed function keys from **F13** to **F20**. When this item is checked, function keys **F11-F20** are mapped onto the control function keys; e.g. **F15** is mapped to **CTRL+F5**, **SHIFT+F13** is mapped to **CTRL+SHIFT+F3**.

VT-52 Escape Sequences

ESC **A**	Cursor up.
ESC **B**	Cursor down.
ESC **C**	Cursor right.
ESC **D**	Cursor left.
ESC **F**	Enter line graphics mode.
ESC **G**	Exit line graphics mode.
ESC **H**	Cursor home.
ESC **I**	Reverse line feed.
ESC **J**	Erase from cursor to end of screen.
ESC **K**	Erase from cursor to end of line.
ESC **V**	Print cursor line.
ESC **]**	Print screen.
ESC **W**	Transparent print mode.
ESC **X**	Cancel transparent print mode.
ESC **^**	Auto print mode.
ESC **_**	Cancel auto print mode.
ESC **=**	Keypad application mode on.
ESC **>**	Keypad application mode off.
ESC **<**	Enter VT220 mode.
ESC **Z**	Identify terminal.
ESC **Y** ***line col***	Cursor position.

PICK PC CONSOLE PROGRAMMING

The Pick Operating System on the IBM PC has its own terminal type for the system console (term type **I**). AccuTerm emulates the Pick PC console thus allowing you to use the same term type codes for all your terminals in a Pick PC environment. The advantage of using this emulation is that it gives you direct access to all the colors and attributes available to the PC.

Operating Modes

ESC ***** **Y** Set the video mode to 80 by 25 monochrome mode. This command is ignored by AccuTerm.

ESC ***** **]** Sets the video mode to 80 by 25 color. This command is ignored by AccuTerm.

GS If the graphics mode is enabled, this command puts the terminal in the Tektronix emulation mode. If the graphics mode is disabled, this command is ignored.

ESC **% ! 0** Enters Tektronix 4100 graphics mode.

Cursor Positioning

BS or ***ESC*** ***** ***HT***
Cursor left. The cursor is moved left one position. If the cursor is at the first column, it is moved to the last column of the previous line.

ESC ***** ***DC3*** Cursor right. Moves the cursor right one position. If the cursor is at the last column, it will wrap to the first column of the next line.

LF Cursor down. Moves the cursor down one row. If the cursor is on the last row of the screen and the terminal is not in protected mode, the screen scrolls up one line.

ESC * LF Cursor up. Moves the cursor up one row. If the cursor is at the top row and the terminal is not in protected mode, the screen is scrolled down one line.

CR Carriage return. Moves the cursor to the leftmost column of the current row.

ESC * STX Cursor home. Moves the cursor to the upper left corner of the screen.

ESC = c r

This command positions the cursor to column ***c*** and row ***r***. ***c*** and ***r*** are the binary cursor positions, numbered from 0 (top row, left column). For example, to position the cursor to column 10 row 10, you would send ***ESC = LF LF*** where ***LF*** is the ASCII representation of character 10.

Erasing and Editing

FF or ***ESC * SOH***

Clears the screen and moves the cursor to the upper left corner of the screen.

ESC * ETX Clears from the cursor position to the end of the screen.

ESC * EOT Clears from the cursor position to the end of the line.

ESC * h Insert character. A blank is inserted at the current cursor position. All characters from the cursor position to the right are moved right one column.

ESC * i Delete character. The character at the current cursor position is deleted. All characters to the right of the cursor are moved left one column and a blank is placed on the last column of the line.

ESC * j Insert line. A blank line is inserted before the current line. The current line and all lines below are moved down one row.

The bottom line of the screen is lost. The cursor is moved to the left most column of the inserted line.

ESC ***** **k** Delete line. The current line is deleted. All lines below the current line are moved up one position. The bottom line is blank. The cursor is moved to the left most column.

Video Attributes

On the PC there are several different ways to set the video attributes. The first is through escape sequences that start and stop individual types like Underline, Reverse, etc. The second is by sending the desired foreground and background colors. One note here, the PC monitor treats the dim attribute as the protected attribute. To change video attributes, you must use the following escape sequences.

ESC ***** ***ENQ*** Start blinking characters.

ESC ***** ***ACK*** Stop blinking characters.

ESC ***** ***BEL*** Start dim (protected) characters.

ESC ***** ***BS*** Stop dim (protected) characters.

ESC ***** ***CR*** Start reverse video characters.

ESC ***** ***SO*** Stop reverse video characters.

ESC ***** ***SI*** Start underlined characters.

ESC ***** ***DLE*** Stop underlined characters.

ESC * n Sets the foreground/background attributes separately where ***n*** is the code from table below.

Color	Background	Bright Foreground	Dim Foreground
White	!	)	9
Yellow	"	*	:
Magenta	#	+	;
Red	$	,	<
Cyan	%	-	=
Green	&	.	>
Blue	'	/	?
Black	(	0	@

Protected Attributes

The Pick PC monitor uses the dim attribute as the protected attribute. When the protect mode is enabled, the cursor is not allowed to move to any positions that have protected characters, the auto scroll mode is turned off, and the clear screen functions clear only the non-protected fields.

ESC * VT Turn on the protected mode.

ESC * FF Turn off the protected mode.

ESC * BEL Start protected (dim) characters.

ESC * BS Stop protected (dim) characters.

Printer Control

ESC* [? 5 i** or ***ESC* * *DC1

Transparent print mode. Characters are printed, but not displayed. When AccuTerm receives this command, it goes into transparent print mode until it receives the ANSI print off sequence below. This command is useful for printing graphics data to the printer. Since AccuTerm requires the specific ANSI cancel command below to exit transparent print mode, there is less chance of a control character interrupting the graphic printing.

ESC* [? 4 i** or ***ESC* * *DC2

Cancel transparent print mode.

MULTIVALUE SERVER OBJECT

When AccuTerm is connected to a MultiValue host, and the host is running a special server program, **FTSERVER** (installed when you install the AccuTerm host programs), other Windows applications (client applications) can request and update data from the host database using the ActiveX `Server` object.

The `Server` object, in conjunction with the **FTSERVER** host program, provides a simple method for performing common host database operations, including reading and writing files, executing commands, calling BASIC subroutines and converting data. The `Server` object can be used in any ActiveX enabled client application, such as Microsoft Word, Excel, and Visual Basic.

When an AccuTerm session is in server mode, it is dedicated to providing MultiValue database services to client programs, and cannot be used as a normal terminal. A "Server Mode" message is displayed while the session is in server mode. When server mode is terminated, normal terminal functions are resumed. To start server mode, enter **FTSERVER** on the Pick TCL command line. To terminate server mode, click the **Cancel** button.

Configuring the Server
After installing the host programs (see the Users Guide for instructions on installing the host programs), use the **FTSETUP** utility to configure the server. You should use **FTSETUP** before you use the server to establish the server name. You can also enable or disable individual services (read, write, convert, execute) using **FTSETUP**.

Accessing the MultiValue Host
To access the MultiValue host, open an AccuTerm session to the desired host, and type **FTSERVER** at the TCL command prompt. AccuTerm should respond by displaying a "Server Mode" status panel identifying the host server name.

Next, in your client application (Word, Excel, VB, etc.) macro environment (Tools ⇨ Macro ⇨ Visual Basic Editor), add a reference to "AccuTerm Pick Server" (Tools ⇨ References). *Note: it is not strictly necessary to add the reference, but doing so allows for automatic parameter type checking and provides a slight performance increase. If you do not add the reference, declare your server object variable as type* ***Object*** *rather than as* ***atPickServer.Server****.* Next, add a module (Insert ⇨ Module). In your Sub or Function, declare a variable for the server object:

```
Dim PickServerObject As atPickServer.Server
```

Next, use the `CreateObject()` function to create an instance of the server object:

```
Set PickServerObject =
      CreateObject("atPickServer.Server")
```

Use the `Connect` method of the server object to establish a connection to the host:

```
If PickServerObject.Connect() Then …
```

The `Connect` method returns True if the connection is successful.

Use the `ReadItem, WriteItem, AddItem, DeleteItem, UnlockItem, OConv, IConv, Execute, CallSub, Download, Upload, Export` or `Import` methods to access the host database. Each of the methods is described in detail in the following topics.

When you are finished using the server, you can disconnect and release the server object:

```
PickServerObject.Disconnect
Set PickServerObject = Nothing
```

If you are going to use the server object multiple times (perhaps from multiple Subs or Functions), you can make the server object variable global (declare it before all Subs or Functions in the module). Then you could write a common function to create the server object and connect to the host:

```
Dim PickServerObject As atPickServer.Server

Private Function ConnectServer() As Boolean
   On Error Resume Next
   If Not PickServerObject Is Nothing Then
      ConnectServer = True
   Else
      Set PickServerObject =
      CreateObject("atPickServer.Server")
      If Not PickServerObject Is Nothing Then
         If PickServerObject.Connect() Then
            ConnectServer = True
            Exit Function
         Else
            Set PickServerObject = Nothing
         End If
      End If
      ConnectServer = False
   End If
End Function
```

All other Subs or Functions which require host database access would first call the `ConnectServer()` function to make sure that services are available:

```
Public Function PickRead(File As String, ID As
      String) As String
   If ConnectServer() Then
      PickRead = PickServerObject.ReadItem(File,
      ID)
   End If
End Function
```

After calling any of the server methods, check the result of the operation by examining the `LastError` property. To get a description of the error, use the `LastErrorMessage` property.

```
If PickServerObject.LastError Then
   MsgBox "Server Error: " &
   PickServerObject.LastErrorMessage
End If
```

Some standard errors returned by the FTSERVER host program are shown below; other codes may be returned from called Pick/BASIC subroutines.

-1	End of file
201	Unable to open file
202	Unable to read item
223	Item already exists
235	Unable to update locked item
260	Unable to read locked item

For further information on using the Pick server, a sample Excel worksheet and Visual Basic application are included in the SAMPLES sub-directory.

Server Functions

`AddItem` method

`Server.AddItem` ***file, ID, data***

Writes a new item to the Pick database. If the item already exists, error 223 is returned.

`CallSub` function

`Result = Server.CallSub(`***SubName*** `[,` ***function*** `[,` ***data***`]] )`

Calls a Pick/BASIC subroutine (***SubName***) passing ***function*** and ***data*** as arguments. The Pick/BASIC subroutine must accept 4 arguments: Function, Data, ErrorCode, ErrorMessage. The returned value is stored in the Data argument before the subroutine returns. If an error occurs, the subroutine must set the ErrorCode argument to a non-zero value, and place the error message text in the ErrorMessage argument.

`Connect` function

```
Result = Server.Connect([ ServerName ])
```

Connects the server object to an AccuTerm session in server mode. If the optional ***ServerName*** is specified, then the object is connected to the specified server; otherwise, the first session which is in server mode is used.

`DeleteItem` method

```
Server.DeleteItem file, ID
```

Deletes an item from the Pick database. If the item is locked by another process, error 235 is returned.

`Disconnect` method

```
Server.Disconnect
```

Disconnects the server object.

`Download` method

```
Server.Download SourceFile, SourceIDs, TargetFolder
  [, Protocol [, Binary [, Overwrite ]]]
```

Uses the AccuTerm File Transfer Utilities to download files from the MultiValue host to the PC. ***SourceFile*** is the host file name; ***SourceIDs*** is a list of item-IDs, an asterisk (*) for all items, or an open parenthesis followed by the name of a saved list. Separate IDs with CR/LF; ***TargetFolder*** is the destination directory. ***Protocol*** is 1 for Kermit (default) or 0 for ASCII protocol. If ***Binary*** is zero (default) attribute marks are translated into CR/LF. ***Overwrite*** is non-zero to allow existing files to be overwritten (default is no overwrite).

`Execute` function

```
Result = Server.Execute(command [, data [,
  capture ]])
```

Executes a TCL ***command*** on the host system. ***Data*** is passed as "stacked input" to the command. If ***capture*** is non-zero, the result of the command is returned, otherwise, the return value is empty.

`Export` method

```
Server.Export SourceFile, SourceIDs, TargetFile,
  TargetFields [, Header [, Explode [, Protocol [,
  Delimiter [, Overwrite ]]]]]
```

Uses the AccuTerm File Transfer Utilties to download a file from the MultiValue host to the PC. ***SourceFile*** is the host file name. ***SourceIDs*** is a list of item-IDs, an asterisk (*) for all items, or an open parenthesis followed by the name of a saved list. Separate IDs with CR/LF; ***TargetFile*** is the destination file. If the destination file has a supported extension (.XLS, .WK1, .WKS, .WB1, .SYM, .DB2, .DBF), the resulting file will be exported in the native format for that file extension. ***TargetFields*** is a list of fields to include in the destination file; use asterisk (*) for all fields. Separate field names with CR/LF. If ***Header*** is non-zero, the first line of the exported file will be a "header". If ***Explode*** is non-zero, single values are repeated for items with multiple values. ***Protocol*** is 1 for Kermit (default) or 0 for ASCII protocol. If the destination file does not have a supported extension, then ***delimiter*** is used to specify Tab (0, default) or Comma (1) delimited ASCII file. ***Overwrite*** is non-zero to allow existing files to be overwritten (default is no overwrite).

`FileExists` function

```
Result = Server.FileExists(file)
```

Returns 1 if the file exists on the host account, or 0 if it does not exist.

`IConv` function

```
Result = Server.IConv(data, code)
```

Performs an "input conversion" on ***data***. Conversion to be performed is ***code***.

`Import` method

```
Server.Import SourceFile, TargetFile, reserved1,
  Fields [, Header [, Skip [, AutoID [,AutoIDPrefix [,
  AutoIDStart [, Protocol [, Delimiter [, Overwrite
  ]]]]]]]
```

Uses the AccuTerm File Transfer Utilities to upload a file from the PC to the MultiValue host. ***SourceFile*** is the PC file name; ***TargetFile*** is the destination file name. If the source file has a supported extension (.XLS, .WK1, .WKS, .WB1, .SYM, .DB2, .DBF), the file will be imported from the native format for that file extension. ***Reserved1*** is reserved for future use and should be passed as a null string. ***Fields*** is a list of fields to be imported (dictionary names or attribute numbers, or asterisk (*) for all fields). Separate field names with CR/LF. If ***Header*** is non-zero, the first

line of the imported file is treated as a "header". ***Skip*** is the number of "header" lines in the source file to skip. If ***AutoID*** is non-zero, then the target item-IDs will be generated by concatenating ***AutoIDPrefix*** to the item sequence starting with ***AutoIDStart***. ***Protocol*** is 1 for Kermit (default) or 0 for ASCII protocol. If the source file does not have a supported extension, then ***delimiter*** is used to specify Tab (0, default) or Comma (1) delimited ASCII file. ***Overwrite*** is non-zero to allow existing items to be overwritten (default is no overwrite).

`IsConnected` function

```
Result = Server.IsConnected()
```

Returns TRUE if server object is connected to the host and the host is running the server application; otherwise returns FALSE.

`ItemExists` function

```
Result = Server.ItemExists(file, ID)
```

Returns 1 if the item exists in the specified file or 0 if the item does not exist or the file cannot be opened.

`OConv` function

```
Result = Server.OConv(data, code)
```

Performs an "output conversion" on ***data***. Conversion to be performed is ***code***.

`ReadItem` function

```
Result = Server.ReadItem(file, ID [, attr [, value [,
  subvalue [, locked ]]]] )
```

Reads an item from the host database and returns the item, attribute, value or sub-value. If the optional ***locked*** parameter is non-zero, the item is left locked after the read (same as BASIC READU statement). Error 260 is returned if the item was already locked by another process.

`Readnext` function

```
Result = Server.Readnext(file, attr [, value [,
  subvalue ]] )
```

Reads the next item from the host database using the current select list and returns the specified attribute, value or sub-value. If the list is exhausted, `LastError` will be set to –1.

`UnlockItem` method

`Server.UnlockItem [` ***file*****`,`** `[,` ***ID*** `]]`

Unlocks an item locked by the ReadItem function. If ***file*** and ***ID*** are null, unlocks all items locked by the process.

`Upload` method

`Server.Upload` ***SourceFolder*****`,`** ***SourceFiles*****`,`** ***TargetFile***
`[,` ***Protocol*** `[,` ***Binary*** `[,` ***Overwrite*** `]]]`

Uses the AccuTerm File Transfer Utilities to upload files from the PC to the MultiValue host. ***SourceFolder*** is the PC directory where the files are uploaded from; ***SourceFiles*** is a list of file names separated by CR/LF. ***TargetFile*** is the destination file. ***Protocol*** is 1 for Kermit (default) or 0 for ASCII protocol. If ***Binary*** is zero (default) attribute marks are translated into CR/LF. ***Overwrite*** is non-zero to allow existing files to be overwritten (default is no overwrite).

`WriteItem` method

`Server.WriteItem` ***file*****`,`** ***ID*****`,`** ***data*** `[,` ***attr*** `[,` ***value*** `[,`
subvalue `[,` ***KeepLocked*** `]]]]`

Writes an item, attribute, value or sub-value to the host database. If the optional ***KeepLocked*** parameter is non-zero, the item is left locked after the writing (same as BASIC WRITEU statement).

Server Properties

`AccountName` property (string)

Returns the name of the account the server object is connected to.

`ErrorMode` property (integer)

Set to 0 to ignore errors, 1 to raise an error to the calling application, 2 to display error message dialog box

`LastError` property (integer)

Returns any error number from the previous operation, or zero if no error.

`LastErrorMessage` property (string)

Returns any error message from the previous operation, or null if no error.

`MDName` property (string)

Returns the file name of the master dictionary (MD or VOC).

`ServerID` property (string)
Returns a unique identifier for the server object.

`ServerName` property (string)
Returns the server name assigned using FTSETUP.

`UnicodeDelimiters` property (integer)
Set to 1 to convert system delimiter characters to Unicode, or 0 to use traditional character values for system delimiters. When Unicode delimiters are specified, AM is U+F8FE, VM is U+F8FD and SVM is U+F8FC.

`UserName` property (string)
Returns the name of the user logged on to the host.

GUI DEVELOPMENT ENVIRONMENT

AccuTerm 2K2 GUI (Graphical User Interface) consists of four components: the AccuTerm GUI Library (BASIC subroutines), the transport mechanism (AccuTerm 2K2), the GUI runtime and the GUI Designer.

AccuTerm GUI applications are based on an *event driven* model. In this model, the application program initializes the GUI environment, then waits for *events*. When the user interacts with a GUI object (clicking a button, for example), an *event* occurs, which the application then processes. This cycle is repeated until the last *event* is processed, at which time the application terminates. Typically, only user actions trigger events. Subroutine calls in the application program generally do not trigger events. The application must be designed to handle arbitrary events in an arbitrary sequence.

The application program interacts with the GUI by calling subroutines from the GUI Library. These routines create GUI objects, set and retrieve the properties (value, color, size, position, font, etc.) of those objects, invoke methods on the objects (reset, close, etc.) and wait for user-initiated events from the objects.

The GUI Designer

The GUI Designer is used to create and maintain GUI projects. The GUI project contains all information required create a GUI application including all of the forms and all of the constituent controls (fields) and set the initial value of most properties. The project record is 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 OK button to create the control (or form).

To move a control, 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.

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 select Properties from the Edit 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.

When the project is executed by the GUI runtime, each item is created in the same order as the items in the project tree. This is also the "tab order" (see the **Tab Order** topic later in this document). 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, select Save from the File menu to save the updated GUI project record back on your host system.

Code Creation and Update

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 Code Editor 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.

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 staed 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.

Updating Code
Once you have generated the base program for a GUI project, you can use Update Code from the Tools menu to check if your program is missing any critical code segments, or if the *event decoder* requires updating. The Update Code dialog will also show you any events that are no longer necessary. 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!*

Use the Update Code *tool frequently to check the status of the event decoder.* ***An out-of-date event decoder is the most common reason a GUI program does not function correctly!***

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 is primarily intended 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.

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

AccuTerm GUI Library Overview

The AccuTerm GUI Library provides a collection of 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 file, and is installed during installation of the AccuTerm host 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 enviroment (ATGUIDELETE, ATGUISHUTDOWN)
Stop
```

The typical event handler will query the GUI application for the values of controls (`ATGUIGETPROP`, `ATGUIGETUPDATES`), validate data, retrieve or update data from the host data base, update values of controls (`ATGUISETPROP`, `ATGUILOADVALUES`), show (`ATGUISHOW`) or hide (`ATGUIHIDE`) forms, activate specific controls or forms (`ATGUIACTIVATE`), 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).

AccuTerm GUI Library Subroutines

The AccuTerm GUI Library routines are organized into the following functional groups:

GUI Environment Functions

ATGUIINIT2	initializes the GUI environment
ATGUISHUTDOWN	terminates the GUI environment
ATGUISUSPEND	temporarily suspends GUI applications
ATGUIRESUME	resumes GUI applications after being suspended

GUI Object Creation Functions

ATGUICREATEAPP	creates an *application* object
ATGUICREATEFORM	creates a *form* object
ATGUICREATEEDIT	creates an *edit* control (text box)
ATGUICREATELABEL	creates a *label* control
ATGUICREATELIST	creates a single-select, multi-select or dropdown *list*
ATGUICREATECOMBO	creates a drop-down *combo box*
ATGUICREATEOPTION	creates an *option group* control (radio buttons)
ATGUICREATECHECK	creates a *check-box* control
ATGUICREATEBUTTON	creates a *command button*
ATGUICREATEGRID	creates a *grid* control
ATGUICREATEPICTURE	creates a *picture box* (container) control
ATGUICREATEFRAME	creates a frame (container) control
ATGUICREATETABGRP	creates a *tab group* – a container for *tab* controls
ATGUICREATETAB	creates a *tab* control
ATGUICREATETREE	creates a *tree* control
ATGUICREATEGAUGE	creates a *progress bar*
ATGUICREATEMENU	creates a *menu*
ATGUICREATETOOLBAR	creates a *toolbar*
ATGUIDELETE	deletes a *control*, *form* or *application*

GUI State Functions

ATGUIACTIVATE	activates a control (set focus)
ATGUISHOW	makes a control or form visible
ATGUIHIDE	makes a control or form invisible
ATGUIENABLE	enables a control or form
ATGUIDISABLE	disables a control or form
ATGUIMOVE	moves or resizes a control
ATGUIGETACTIVE	returns the active application, form and control

GUI Property Functions

ATGUISETPROP	set the value of a property of a GUI object
ATGUIGETPROP	returns the value of a property of a GUI object
ATGUISETPROPS	set 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 object on a form
ATGUISETITEMPROP	set 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 Value property of multiple GUI objects
ATGUIGETVALUES	returns the Value property of multiple GUI objects
ATGUIGETUPDATES	returns Value property from updated GUI objects
ATGUIRESET	resets the Value property of all GUI objects on a form to its default value
ATGUICLEAR	clears the Value property of any GUI object on a form
ATGUIINSERT	insets a blank row in a list, combo-box or grid
ATGUIINSERTITEMS	inserts one or more rows into a list, combo-box or grid
ATGUIREMOVE	removes one row from a list, combo-box or grid
ATGUIREMOVEITEMS	removes one or more rows from a list, combo-box or grid

GUI Event Processing Functions

ATGUIWAITEVENT	waits for the next user-interface event
ATGUICHECKEVENT	waits for the next event or timeout
ATGUIPOSTEVENT	simulates an event

GUI Macro Functions

ATGUIBEGINMACRO	begins recording a macro
ATGUIENDMACRO	ends macro recording
ATGUIRUNMACRO	plays a recorded macro or loads a project template created by the GUI Designer

GUI Utility Functions

ATGUIPRINT2	prints the form or displays the Print Setup dialog
ATGUIHELP	displays help page in dedicated browser window
ATGUIMSGBOX	displays a message
ATGUIINPUTBOX	prompts for single line of text
ATGUIFILEDIALOG	displays the standard Open, Save As or Browse for Folder dialog
ATGUICOLORDIALOG	displays the standard color dialog
ATGUIFONTDIALOG	displays the standard font dialog

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. The GUISTATE variable is used internally by the GUI routines to maintain state information while a GUI application is running. This variable must not be altered, and must be passed all other GUI functions.

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 1.3.
GUISTATE	internal use

Output arguments:

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

If an error occurs, GUIERRORS<1> will be non-zero. See the **Errors** topic for details.

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

ATGUISUSPEND
ATGUIRESUME

The ATGUISUSPEND routine suspends the currently running GUI applications and allows normal terminal function. While a GUI application is suspended, it is hidden. Call ATGUIRESUME to resume a suspended GUI application. The GUISTATE variable must be maintained between the ATGUISUSPEND and ATGUIRESUME calls.

Calling syntax:

```
CALL ATGUISUSPEND(GUIERRORS, GUISTATE)

CALL ATGUIRESUME(GUIERRORS, GUISTATE)
```

Input arguments:

GUISTATE	internal use

Output arguments:

GUISTATE	internal use
GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see the **Errors** topic for details)

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 element 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 the **Errors** topic for details)

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 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 application 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.
GPAUTOSEL	non-zero if text fields are automatically selected when activated
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)
GPCOPYRIGHT	application copyright notice (shown in the About box)
GPAUTHOR	application author (shown in the About box)

GPVERSION	application version (shown in the About box)
GPLOGO	filename or URL of logo bitmap 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 seconds. 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	set to 1 to automatically select the text in an edit control when it is activated by the tab key
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	window state of the MDI main window: 0 = normal, 1 = minimized, 2 = maximized (MDI app only)

Events supported by this item:

GEQUIT	the GUI application has terminated – end of event processing
GEACTIVATE	a form that belongs to the application has been activated. GUIARGS<1,1> is the ID of the application being deactivated.

GEDEACTIVATE	a different application has been activated. GUIARGS<1,1> is the ID of the application being activated.
GERESIZE	the user has resized the main application window. The new width is GUIARGS<1,1>, height is GUIARGS<1,2> and window state is GUIARGS<1,3>
GEHELP	the user has requested help. GUIARGS<1,1> is 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> is the topic ID and GUIARGS<1,3> is the ID of the active control. GUIARGS<1,4> is the "post data" from the html page. The host program should call the ATGUIHELP routine to display the requested help page.

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 not visible; call ATGUISHOW to show the *form*.

Calling syntax:

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

Input arguments:

APPID	identifies which application 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 the **Errors** topic for details)

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	window state of the form window: 0 = normal, 1 = minimized, 2 = maximized

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 user has resized the form window. The new width is GUIARGS<1,1>, height is GUIARGS<1,2> and window state is GUIARGS<1,3>. *Note: fixed size forms may receive this event if the user minimizes or restores the form to the taskbar.*

ATGUICREATEEDIT

The ATGUICREATEEDIT routine creates an *edit*, or text box, control. The type of control may be single line, multi line or password. 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. 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	0 = single line, 1 = multi-line, 2 = password
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 the **Errors** topic for details)

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
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 = 3D 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)
GPDEFVAL	default value of control (also sets current value)
GPVALUE	value of control (separate multiple lines using value marks)

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 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> is the ID of the control triggering the event and GUIARGS<2> is the 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.
GEDEACTIVATE	the control is no longer the active control. GUIARGS<1,1> is the ID of the control being activated.

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

You can include an ampersand (**&**) in the label caption to use as a "hot key" to activate the next control in the project. When the user presses the letter following the ampersand while holding the **ALT** key, the next control (usually an *edit* control) will be activated.

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)
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 the **Errors** topic for details)

Properties supported by this item:

GPFORECOLOR	foreground (text) color (default is form's FORECOLOR)
GPBACKCOLOR	background color (default 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
GPENABLED	non-zero if cgfontrol is enabled
GPVISIBLE	non-zero if control is visible
GPBORDER	0 = no border (default); 1 = flat border; 2 = 3D border
GPALIGN	0 = left, 1 = right, 2 = center
GPSTYLE	0 = single line, 1 = multi-line
GPDEFVAL	default value of control (also sets current value)
GPVALUE	value of control (separate multiple lines using value 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
GECONTEXT	the user clicked the right mouse button on this control
GEDBLCLICK	the user double-clicked the left mouse button on this control

ATGUICREATELIST

The ATGUICREATELIST routine creates a *list* control. The style of control may be single or multi selection, or a drop-down list. A *list* control is used to select one or more items from a list of items. 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. 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; otherwise the height is the height of the displayed list.

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	0 = single selection, 1 = multiple selections, 2 = drop-down list
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 the **Errors** topic for details)

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
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 = 3D border (default)
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.
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.
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> is the ID of the control triggering the event and GUIARGS<2> is the 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.

ATGUICREATECOMBO

The ATGUICREATECOMBO routine creates a *combo box* control. A *combo box* control is a combination of *edit* and *list* controls. A *combo box* control is used to select an item from a list of items, or enter text directly. The *value* of a *combo box* control is the text string of the edit portion of the control. The *combo box* 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.

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 the **Errors** topic for details)

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
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 = 3D 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.
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	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.
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> is the ID of the control triggering the event and GUIARGS<2> is the 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 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.

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

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).

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
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 the **Errors** topic for details)

Properties supported by this item:

GPFORECOLOR	foreground (text) color (default is form's FORECOLOR)
GPBACKCOLOR	background color (default 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
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 = 3D border (default)
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.
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> is the ID of the control triggering the event and GUIARGS<2> is the 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.

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

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.

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	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 the **Errors** topic for details)

Properties supported by this item:

GPFORECOLOR	foreground (text) color (default is form's FORECOLOR)
GPBACKCOLOR	background color (default 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
GPENABLED	non-zero if control is enabled
GPVISIBLE	non-zero if control is visible
GPCAPTION	caption text. Multiple lines are separated by value marks.
GPHELPID	help topic ID or URL
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> is the ID of the control triggering the event and GUIARGS<2> is the modified value of the control.
GECLICK	the user toggled the state of this control by clicking the left mouse button or by pressing the spacebar when the control is active – GUIARGS<1> is the new state
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.

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

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 RTNEQTAB 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 RTNEQTAB 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.

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 the **Errors** topic for details)

Properties supported by this item:

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
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	0 = normal, 1 = OK button, 2 = Cancel button
GPHELPID	help topic ID or URL
GPPICTURE	optional picture filename or URL for graphical buttons (do not use GPCAPTION for graphical buttons)

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 or pressed the spacebar when the control is active
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.

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. A label field can have an ellipsis button which triggers a special Ellipsis event when clicked. In addition, the lists displayed for drop-down list and drop-down combo fields 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. 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).

Users navigate a *grid* by using the cursor keys, **TAB** key (and **ENTER** key if the RTNEQTAB 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.

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 the **Errors** topic for details)

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
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 = 3D 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
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	0 = label, 1 = edit, 2 = check box, 3 = drop-down list, 4 = drop-down combo, 5 = label with ellipsis button. *Note: only label, edit and check box are supported for non-editable grids.*
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.
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.
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.
GPDEFVAL	default value of control (also sets the current value).
GPVALUE	value of 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.
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.
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.
GECOLCLICK	the user clicked the left mouse button on the column heading. GUIARGS<1,1> is the column number clicked.
GEELLIPSIS	the user clicked the ellipsis button (…) in a cell. GUIARGS<1,1> is the column number and GUIARGS<1,2> is the row number of the cell where the ellipsis 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> is the column number and GUIARGS<1,2> is the row number of the changed cell. GUIARGS<2> is the new value of the 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> is the ID of the control triggering the event, GUIARGS<2> is the 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> is the column number and GUIARGS<1,2> is the row number of the cell being validated. GUIARGS<2> is the new cell value. Only editable grids fire this 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> is the row number being validated. GUIARGS<2> contains the values of all of the row's cells separated by sub-value 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). The new column number is GUIARGS<1,1> and the new row number is GUIARGS<1,2>. The previous column number is

	GUIARGS<1,3> and previous row number is GUIARGS<1,4>. If the grid has just become the active control, the previous column and row arguments are NULL and the ID of the previously active control is GUIARGS<1,5>. Otherwise GUIARGS<1,5> is NULL.
GEACTIVATEROW	the user has moved to a new row (or the grid has just become the active control). The new row number is GUIARGS<1,1>. The previous row number is GUIARGS<1,2>. If the grid has just become the active control, the previous row argument is NULL and the ID of the previously active control is GUIARGS<1,3>. Otherwise GUIARGS<1,3> 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). The previous column number is GUIARGS<1,1> and the previous row number is GUIARGS<1,2>. The new column number is GUIARGS<1,3> and new row number is GUIARGS<1,4>. If the grid has just been deactivated, the new column and row arguments are NULL and the ID of the newly activated control is GUIARGS<1,5>. Otherwise GUIARGS<1,5> is NULL.

GEDEACTIVATEROW	the user has moved to a new row (or the grid has just been deactivated). The previous row number is GUIARGS<1,1>. The new row number is GUIARGS<1,2>. If the grid has just been deactivated, the new row argument is NULL and the ID of the newly activated control is GUIARGS<1,3>. Otherwise GUIARGS<1,3> is NULL.

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
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)
STYLE	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, .jpg, .gif, .wmf and .emf.
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 the **Errors** topic for details)

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 = 3D border (default)
GPSTYLE	0 = scale image to control, 1 = scale image preserving aspect ratio, 2 = resize control to fit image
GPPICTURE	filename or URL of picture. Picture formats supported include .bmp, .jpg, .gif, .wmf and .emf.
GPSTATUS	returns status information about the form: if COL = 2, returns a multi-valued list of child objects (controls which have the picture object as their parent).

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

ATGUICREATEFRAME

The ATGUICREATEFRAME 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.

You can include an ampersand (**&**) in the frame caption to use as a "hot key" to activate the first control inside the frame. When the user presses the letter following the ampersand while holding the **ALT** key, the first control inside the frame will be activated.

Calling syntax:

```
CALL ATGUICREATEFRAME(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	caption text
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 the **Errors** topic for details)

Properties supported by this item:

GPFORECOLOR	foreground (text) color (default is form's FORECOLOR)
GPBACKCOLOR	background color (default 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
GPENABLED	non-zero if control is enabled
GPVISIBLE	non-zero if control is visible
GPBORDER	0 = no border; 1 = flat border; 2 = 3D border (default)
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 frame as their parent).

Events supported by this item (sum the desired events to form the EVENTMASK argument): none

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.

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 the **Errors** topic for details)

Properties supported by this item:

GPFORECOLOR	foreground (text) color (default is form's FORECOLOR)
GPBACKCOLOR	background color (default 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
GPENABLED	non-zero if control is enabled
GPVISIBLE	non-zero if control is visible
GPSTYLE	0 = single row of tabs, 1 = multiple rows of 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).
GPVALUE	index of the 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.

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.

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 a "parent" container control, which must be a *tab group* 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 the **Errors** topic for details)

Properties supported by this item:

GPFORECOLOR	foreground (text) color (default is tab group's FORECOLOR)
GPBACKCOLOR	background color (default is tab group's 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).

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.

ATGUICREATETREE

The ATGUICREATETREE routine creates a *tree* control. A *tree* control displays a hierarchical structure 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 backslash (\ ) characters.

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, which must be a *tab group* control
CTRLID	control identifier
EVENTMASK	list of events this control responds to
STYLE	0 = text only, 1 = text & sign, 2 = text & small icon, 3 = text & small icon & sign, 4 = text & large icon, 5 = 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 the **Errors** topic for details)

Node item record format:

subvalue 1	node ID (unique within sibling nodes)
subvalue 2	node level (level 1 = top level node)
subvalue 3	node description
subvalue 4	node icon (filename or URL)
subvalue 5	node state (0=collapsed, 1=expanded)
subvalue 6	node hint (tooltip) text (single line only)

The level of each node item must be either: a) the same as the level ofthe previous node; or b) one more than the level of the previous node; or c) less than the previous node, but greater than zero.

Properties supported by this item:

GPFORECOLOR	foreground (text) color (default is WindowText)
GPBACKCOLOR	background color (default 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
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 = 3D border (default)
GPSTYLE	0 = text only, 1 = text & sign, 2 = text & small icon, 3 = text & small icon & sign, 4 = text & large icon, 5 = 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.
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> is the ID of the control triggering the event and GUIARGS<2> is the 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. The affected node ID is returned in GUIARGS<1,1> and the new state in GUIARGS<1,2>.
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.

ATGUICREATEGAUGE

The ATGUICREATEGAUGE routine creates a *gauge* control, also know as a *progress bar*. A *gauge* control is used 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
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 = segmented progress bar, 1 = smooth progress bar
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 the **Errors** topic for details)

Properties supported by this item:

GPFORECOLOR	foreground (text) color (default is form's FORECOLOR)
GPBACKCOLOR	background color (default 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
GPENABLED	non-zero if control is enabled
GPVISIBLE	non-zero if control is visible
GPBORDER	0 = no border; 1 = flat border; 2 = 3D border (default)
GPDEFVAL	default value of control (also sets the current value)
GPVALUE	value of control (0 to 100)

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

No events are defined for the *Gauge* control

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*. The *menu* structure is a two-dimensional structure. 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 created in a nested structure. Items on the form main top-level menu are at level zero. Items in a drop-down menu, under a main menu item, are level 1, etc. When constructing the menu structure, the first item should be at level 0, followed by all level 1 items under the level 0 item. Level 2 items may follow the level 1 item they are under, etc.

When a menu is created on 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 the **Errors** topic for details)

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 (level 1 = top level menu, level 2 = dropdown menu, level 3 = sub-menu, etc.), main menus start at level 1, popup (context) menus start at level 1.
subvalue 3	menu item caption
subvalue 4	optional menu item picture (either 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. An individual item can be accessed by ATGUIGETPROP and ATGUISETPROP by specifying non-zero ROW argument.

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.

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
ITEMS	toolbar item records separated by value marks. Each toolbar item record contains 8 fields separated by subvalue marks.

Output arguments:

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

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	tooltip help text
subvalue 4	button picture (either 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 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 button visible state.
GPALIGN	0 = floating, 1 = top, 2 = bottom, 3 = left, 4 = right
GPSTYLE	0 = small icons, 1 = large icons
GPICON	use ATGUIGETITEMPROP and ATGUISETITEMPROP to access

	individual menu item icon filename or URL.
GPITEMS	toolbar item records separated by value marks. Each toolbar item record contains 8 fields separated by subvalue marks. An individual item can be accessed by ATGUIGETPROP and ATGUISETPROP by specifying non-zero ROW argument.

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.

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 the **Errors** topic for details)

ATGUIACTIVATE

The ATGUIACTIVATE function is called to 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 the **Errors** topic for details)

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 the **Errors** topic for details)

Note: contrary to normal design principles, showing a Form may trigger Activate and Deactivate events for the form that becomes active or inactive. Activate, Deactivate and Validate events may also occur for the control on the form that is deactivated, and the control on the form that becomes active.

ATGUIHIDE

The ATGUIHIDE routine makes a form or control 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 the **Errors** topic for details)

Note: contrary to normal design principles, hiding a Form may trigger Activate and Deactivate events for the form that becomes active or inactive. Activate, Deactivate and Validate events may also occur for the control on the form that is deactivated, and the control on the form that becomes active.

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 the **Errors** topic for details)

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 the **Errors** topic for details)

ATGUIMOVE

The ATGUIMOVE routine moves or resizes a control or form.

Calling syntax:

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

Input arguments:

APPID	application identifier
FORMID	form identifier
CTLID	control identifier
LEFT	new left position (null to keep current left)
TOP	new top position (null to keep current top)
WIDTH	new width (null to keep current width)
HEIGHT	new height (null to keep current height)

Output arguments:

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

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 needs to 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 the **Errors** topic for details)

ATGUISETPROP

The ATGUISETPROP routine sets the value of a property of a GUI *application, form* or *control.* See the **Standard Properties** topic for a description of the standard properties. See the individual object creation function for information on the properties supported by each object type.

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)
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 the **Errors** topic for details)

ATGUIGETPROP

The ATGUIGETPROP routine retrieves the value of a property of a GUI *application, form* or *control.* See the **Standard Properties** topic for a description of the standard properties. See the individual object creation function for information on the properties supported by each object type.

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)
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 the **Errors** topic for details)

ATGUISETPROPS

The ATGUISETPROPS routine is called to set 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 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 the **Errors** topic for details)

ATGUIGETPROPS

The ATGUIGETPROPS routine is called to retrieve one or more properties of one or more controls. This function is like calling ATGUIGETPROP(…, …, …) multiple times.

Calling syntax:

```
CALL ATGUIGETPROPSS(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 and multiple columns within a row are separated by subvalue marks.
GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see the **Errors** topic for details)

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 and Toolbar controls. See the individual object creation function for information on the properties supported by each object type.

Calling syntax:

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

Input arguments:

APPID	application identifier
FORMID	form identifier
CTRLID	control identifier
ITMID	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 the **Errors** topic for details)

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 and Toolbar controls. See the individual object creation function for information on the properties supported by each object type.

Calling syntax:

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

Input arguments:

APPID	application identifier
FORMID	form identifier
CTRLID	control identifier
ITMID	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 the **Errors** topic for details)

ATGUILOADVALUES

The ATGUILOADVALUES routine is called to load a set of values onto a GUI *form*. This function is like calling ATGUISETPROP(…, GPVALUE, …) for each control on a 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 the **Errors** topic for details)

ATGUIGETVALUES

The ATGUIGETVALUES routine is called to retrieve a set of values from a GUI *form*. This function is like calling ATGUIGETPROP(…, GPVALUE, …) for each control on a 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 the **Errors** topic for details)

ATGUIGETUPDATES

The ATGUIGETUPDATES routine is called to retrieve all updated values from a GUI form. This function is similar to ATGUIGETVALUES except that instead of passing a list of control IDs, the IDs of updated controls are returned along with their Values.

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 the **Errors** topic for details)

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, or the value assigned by setting the GPDEFVAL property.

Note: ATGUIRESET doees 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 the **Errors** topic for details)

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 the **Errors** topic for details)

ATGUIINSERT

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

Calling syntax:

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

Input arguments:

APPID	application identifier
FORMID	form identifier
CTLID	control identifier
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 the **Errors** topic for details)

ATGUIINSERTITEMS

The ATGUIINSERTITEMS routine inserts one or more new items (rows) into a *list*, *combo box* or *grid* control. The row to be inserted before is specified. Specify zero to add a 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
ROW	row to insert before; zero to add new row 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 the **Errors** topic for details)

ATGUIREMOVE

The ATGUIREMOVE routine deletes a row from a *list, combo box* or *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
ROW	row to delete

Output arguments:

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

ATGUIREMOVEITEMS

The ATGUIREMOVEITEMS routine deletes one or more rows from a *list*, *combo box* or *grid* control. The row to be deleted and number of 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
ROW	first row to delete
COUNT	number of rows to delete

Output arguments:

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

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,
      GUIEVT, GUIARGS, GUIERRORS, GUISTATE)
```

Output arguments:

APPID	application identifier
FORMID	form identifier
CTRLID	control (or menu) identifier which triggered the event
GUIEVT	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)

Output arguments (both):

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

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, GUIEVT, GUIARGS, GUIERRORS,
     GUISTATE)
```

Output arguments:

TIMEOUT	timeout value in milliseconds
APPID	application identifier
FORMID	form identifier
CTRLID	control (or menu) identifier which triggered the event
GUIEVT	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)

Output arguments (both):

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

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,
      GUIEVT, GUIARGS, POSITION, GUIERRORS,
      GUISTATE)
```

Output arguments:

APPID	application identifier
FORMID	form identifier
CTRLID	control (or menu) identifier which triggered the event
GUIEVT	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 the **Errors** topic for details)

ATGUIBEGINMACRO
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 the **Errors** topic for details)

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 substituions (reserved for future use)

Output arguments:

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

ATGUIPRINT2

The ATGUIPRINT2 routine prints a form or displays the Print Setup dialog. To access printer properties, please see the **Printer object** topic in the **Global Objects** section.

Calling syntax:

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

Input arguments:

APPID	application identifier
FORMID	form identifier
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 cancelled the print dialog, GUIERRORS<1> = 1, and GUIERRORS<2,5> = GRCANCEL.

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 ("NEWUSER") in GUIARGS<1,2>.

The final help scheme (help type 2) available for AccuTerm GUI applications uses a web server to deliver 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, 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 (hep 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 the **Errors** topic for details)

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.

Calling syntax:

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

Input arguments:

PRMPT	custom prompt message (multiple lines as OK; separate lines with subvalue marks)
CAPTION	dialog box caption text
STYLE	specifies which icon to display in message box: 0 = no icon, 1 = "**X**", 2 = "**!**", 3 = "**?**", 4 = "**i**"
BUTTONS	specified which buttons to include in message box: 0 = OK only 1 = OK/Cancel 2 = Abort/Retry/Ignore 3 = Yes/No/Cancel 4 = Yes/No 5 = Retry/Cancel add one of the following to specify which button is the “default” 100 = first button 200 = second button 300 = third button
HELPID	topic in help file to access when user presses the **F1** key

Output arguments:

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

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 use can enter a single string response, and either click 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 as 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

Output arguments:

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

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” of “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	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 the **Errors** topic for details)

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)

ATGUIFONTDIALOG

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

Calling syntax:

```
CALL ATGUIFONTDIALOG(FONTNAME, FONTSIZE,
      FONTBOLD, FONTITALIC, 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 sttyle (0=normal, 1=italic)

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 sttyle (0=normal, 1=italic)
GUIERRORS	GUIERRORS<1> is non-zero if errors have occurred (see the **Errors** topic for details)

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... contants 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. Fforms 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
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 leading 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.

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.

GPSTATUS	returns status information about the GUI runtime environment: if COL = -1, this property returns (or sets) the GUI status message text. If COL = 0, it returns the number of visible forms of all application objects; if COL = 1, it returns the ID of the currently active control; if COL = 2, it returns a multi-valued list of application objects. The returned list can be restricted to return only MDI application objects by calling ATGUIGETPROP with ROW = 1 or only SDI application object if ROW = 2, otherwise, if ROW = 0, all application objects will be included in the list.

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.

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 ROW argument; to return the useable height, pass 1 in ROW.

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.

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.

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 error 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

Tab Order and the Caption Property

Each form has a "tab order", which is the order in which control 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.

In the GUI Designer, you can adjust the tab order by dragging and dropping the controls in the project tree.

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.

Multiple Application Model

AccuTerm 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 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 "Cusotm 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.

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\Atwin\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 AccuSoft; 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 Tempalte 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.

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 store in the atguisvr.ini file, in the …\Program Files\Atwin folder. A sample atguisvr.ini file is provided in the …\Program Files\Atwin\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 AccuSoft for distribution with future releases of AccuTerm.

OBJECT BRIDGE

Object Bridge is a Windows Object extension (ActiveX Automation) for BASIC programs. This allows a BASIC program to manipulate nearly any (OLE) Automation compatible object on the Windows machine. Examples of Automation objects include Word documents, Excel spreadsheets, and any public objects implemented using Visual Basic.

Object Bridge consists of three components: a BASIC interface implemented as a set of callable subroutines, a "remote procedure call" (RPC) conduit (AccuTerm 2K2 session) and a Windows object manager (installed with AccuTerm 2K2).

Object Bridge allows a BASIC program to create and destroy instances of Windows objects, set and retrieve property values from an object, invoke methods (subroutine calls and functions) on an object and process events produced by an object. The BASIC subroutines which provide this functionality are described below.

To use Object Bridge effectively, you may need some reference material in order to understand the object model of the application you are trying to control (Word, Excel, VB, etc). Also, the Object Browser tool available in Word or Excel (Macro -> Visual Basic Editor) or in the VB IDE is very useful.

The basic idea is this:

1) Initialize the Object Bridge environment
2) Create an object
3) Set property values, get property values or call subruotines or functions (invoke methods) on the object
4) Respond to desired events produced by the object
5) Release the object

Installing Object Bridge

Object Bridge may be installed when the AccuTerm MultiValue host programs are installed. If ObjectBridge is not installed at this time, you can use the `LOAD-ACCUTERM-OBJBP` utility (which is installed with the host

file transfer programs) to install ObjectBridge at a later time. You will need to create the ObjectBridge program file before running LOAD-ACCUTERM-OBJBP. We recommend OBJBP, but any file name will work.

Some sample programs are included in the SAMPLES folder, can also be installed with the ObjectBridge programs.

Initializing the Object Bridge Environment

```
ATINITOBJMGR(ERRMSG, OPTS)
```

Call this subroutine before using any other Object Bridge routine. This verifies that a compatible version of AccuTerm 2K2 is connected, and initializes some important state information in the OPTS variable. You must maintain and pass the OPTS variable to all other Object Bridge routines.

If any of the Object Bridge routines encounter an error, the ERRMSG arugment will be set to indicate the error, otherwise it will be null. Sometimes you can ignore an error, such as when attempting to set a property to an invalid value; other times you should call ATRESETOBJMGR to release all your objects and terminate the program.

Reset the Object Bridge Environment

```
ATRESETOBJMGR
```

This routine may be called if you want to release all objects and clean up the Windows object manager. This is useful if an error occurs and you want to abort all further use of any objects. You do not need to call ATRELEASEOBJECT if you call ATRESETOBJMGR.

Creating an Object

```
ATCREATEOBJECT(CLSID, OBJID, ERRMSG, OPTS)
```

This routine will attempt to create an object of the desired class (CLSID). Use the returned OBJID to refer to this object. When you no longer require access to the object, call ATRELEASEOBJECT passing the same OBJID, otherwise the object remains present in Windows memory.

Referencing an Existing Object

```
ATGETOBJECT(OBJNAME, CLSID, OBJID, ERRMSG,
OPTS)
```

This routine will attempt to get a reference to an existing object of the desired class (CLSID) and name (OBJNAME). Use the returned OBJID to refer to this object. When you no longer require access to the object, call ATRELEASEOBJECT passing the same OBJID, otherwise the object remains present in Windows memory.

Releasing an Object

```
ATRELEASEOBJECT(OBJID, ERRMSG, OPTS)
```

This routine releases the specified object, as well as any objects "derived" from the specified object (OBJID). A derived object may be an object returned by a call to ATGETPROPERTY or ATINVOKEMETHOD when the return value is itself another object.

Setting Object Properties

```
ATSETPROPERTY(OBJID, PROPNAMES, PROPVALUES,
ERRMSG, OPTS)
```

Use this routine to get the values of one or more properties of an object. Specify the property names you desire in the PROPNAMES argument, separated by attribute marks. The corresponding property values to set are passed in the PROPVALUES argument, also separated by attribute marks. If an error occurs, the error message will be in the corresponding attribute of the ERRMSG argument.

Some properties are actually "array properties". If you need to get the value of an array property, concatenate the property name with a sub-value mark followed by the index value.

Getting Object Properties

```
ATGETPROPERTY(OBJID, PROPNAMES, PROPVALUES,
ERRMSG, OPTS)
```

Use this routine to get the values of one or more properties of an object. Specify the property names you desire in the PROPNAMES argument, separated by attribute marks. The corresponding property values are returned in the PROPVALUES argument, also separated by attribute marks. If an error occurs, the error message will be in the corresponding attribute of the ERRMSG argument.

Some properties are actually "array properties". If you need to get the value of an array property, concatenate the property name with a sub-value mark followed by the index value.

Some properties are actually objects themselves. This is a "derived" object. In this case, you can use the returned property value as an object ID in other Object Bridge calls. When the object which returned the "derived" object is released, all of its derived objects will also be released. Releasing the derived object will not release its creator.

Invoking Methods (calling subroutines and functions)

```
ATINVOKEMETHOD(OBJID, METHNAMES,
METHARGS,RESULT, ERRMSG, OPTS)
```

Call this routine to invoke a "method" on the object. A method is a subroutine or function. A subroutine will not return a value (RESULT will be null). A function will return a result in the RESULT argument. Multiple methods may be called. Separate method names in the METHNAMES argument by attribute marks. If a method requires arguments, the arguments are passed in the METHARGS parameter, with arguments for each method separated by attribute marks. Within the method arguments for one method, multiple arguments are separated by value marks. Results from multiple methods are returned in the RESULTS argument separated by attribute marks.

Some methods return objects. In this case, you can use the returned value as an object ID in other Object Bridge calls. When the object which returned the "derived" object is released, all of its derived objects will also be released. Releasing the derived object will not release its creator.

Enabling Event Processing

```
ATSETEVENT(OBJID, EVENTNAMES, ENABLE, ERRMSG,
OPTS)
```

Many objects support events, which notify the object's owner of certain actions, such as closing the program or clicking a button. Object Bridge only notifies the Pick application of those events which have been enabled by calling ATSETEVENT. Specify the event names in the EVENTNAMES arguments, separated by attribute marks. Specify whether the event is enabled (1) or disabled (0) in the corresponding ENABLE argument, also separated by attribute marks.

Polling for Events

```
ATGETEVENT(OBJID, EVENTNAME, ARGS, TIMEOUT,
ERRMSG, OPTS)
```

Call this routine to wait for an event from any objects for which you have enabled events. You do not pass OBJID to this routine, it is passed back to your program when an event occurs so you can identify the source of the event. EVENTNAME and ARGS are also passed back to your routine. Some events allow you to modify one or more of their arguments, for example, to cancel an operation. You must preserve OBJID, EVENTNAME and ARGS (except for modified values in ARGS) when processing the event so they can be passed to ATENDEVENT when you complete processing the event. Event argument values are separated by value marks. You must always call ATENDEVENT after processing the event. You can call other routines in between, but no more events can be processed, and the Windows object may be roadblocked until you call ATENDEVENT.

If you want ATGETEVENT to return to your program even if an event did not occur, you can specify the number of milliseconds to wait for the event in the TIMEOUT parameter. Set TIMEOUT to zero to wait forever for the event.

Completing Event Processing

```
ATENDEVENT(OBJID, EVENTNAME, ARGS, ERRMSG,
OPTS)
```

Call this routine after handling an event. Pass back the `OBJID`, `EVENTNAME` and `ARGS` as they were returned by the `ATGETEVENT` routine, except for any argument values you have modified.

You must call this routine after processing an event returned by `ATGETEVENT`, otherwise no other events may be processed and the Windows application which implements the object may not be able to terminate.

MOUSE SUPPORT

Custom Mouse Action
AccuTerm supports the use of a Mouse Pattern Table. The Mouse Pattern Table associates patterns of text on the screen with response strings. The response strings can either be sent to the host or executed as macro commands. Each entry specifies a pattern, mouse button, click response and double click response.

AccuTerm's host-controlled mouse functions override the Mouse Pattern Table, and if the host has enabled the mouse using the ***ESC STX*** **1** or ***ESC STX*** **2** command, then the Mouse Pattern Table functions are disabled until the host disables the mouse using the ***ESC STX*** **0** command.

The Mouse Pattern Table is specified in the Advanced settings on the Misc tab of the Settings dialog.

The Mouse Pattern Table file is a tab delimited text file (ANSI character set). Each line in the file is an entry in the Mouse Pattern Table. The default directory for this file is the ATWIN home directory.

Note: the Tab delimiters are important in this file. Using a word processor like Microsoft Word with formatting marks visible is recommended!

Besides the ability to read the Mouse Pattern Table from a file, the table can also be downloaded from the host, and therefore, changed on the fly as applications have different mouse requirements. To download the table, use the following AccuTerm private escape sequence:

ESC STX **h** ***CR***	clears the Mouse Pattern Table
ESC STX **h** ***entry CR***	adds ***entry*** to the Mouse Pattern Table

Each entry is made up of four fields separated by tab (HT) characters:

button HT pattern HT click HT dblclk

The Mouse Pattern Table can contain up to 128 entries. Each entry specifies the mouse button desired (***button***), the pattern (***pattern***), the response to a click (***click***) and the response to a double click (***dblclk***). The fields of each entry are separated by a tab character. The mouse button is a single digit: **1** = left, **2** = right, **3** = middle. Pattern and response strings are described below.

Patterns

The pattern field is a "regular expression", similar to the regular expressions used by the Unix **grep** command:

.	matches any character
^	beginning of line
$	end of line
[*list*]	any characters in ***list*** (***list*** can include a range of characters such as **0-9**
[^*list*]	any characters not in ***list***
:A	any alpha character
:D	digit (0 - 9)
:N	any alpha or numeric character
?	zero or one of the preceding item
*****	zero or more of the preceding item
+	one or more of the preceding item
{	marker for the beginning of the word to return if the pattern matches
}	marker for the end of the word to return if the pattern matches
****	causes the next character in the pattern string to be used literally; for example, **\.** matches a dot
any other char	matches itself

Responses

Reponses are encoded using the function key format (see **Keyboard Settings**). Just like function keys, if the response is a string enclosed in square brackets (**[ ]**), the response is a "macro" command, and it is executed, rather than being sent to the host (see **Scripting**). Note: if the macro begins with an asterisk (*****), it causes a “context menu” to be displayed, rather than executing as a macro. This can be used with a

customized popup menu, described in the next section. For example, a response of **[*reports]** will open the popup menu named “reports”.

Certain special tokens are permitted in the response string which are useful for returning information to the host, or for passing as arguments to a macro command. These special tokens are:

%WORD%	replaced by the "word" which matched the pattern.
%COL%	replaced by the zero-based text column where the mouse was clicked.
%ROW%	replaced by the zero-based text row where the mouse was clicked.
%BEG%	replaced by the zero-based text column where the matched text begins.
%END%	replaced by the zero-based text column where the matched text ends.
%MOVE%	replaced by the cursor movement commands required to position the cursor to the location of the mouse click.
%%	replaced by the percent (%) character.

APPENDIX A

Wyse Tables

Wyse Cursor Address Table

Row or column	Code	Row or column	Code	Row or column	Code
1	*SPACE*	33	@	65	`
2	!	34	A	66	a
3	"	35	B	67	b
4	#	36	C	68	c
5	$	37	D	69	d
6	%	38	E	70	e
7	&	39	F	71	f
8	'	40	G	72	g
9	(	41	H	73	h
10	)	42	I	74	i
11	*	43	J	75	j
12	+	44	K	76	k
13	,	45	L	77	l
14	-	46	M	78	m
15	.	47	N	79	n
16	/	48	O	80	o
17	0	49	P	81	p
18	1	50	Q	82	q
19	2	51	R	83	r
20	3	52	S	84	s
21	4	53	T	85	t
22	5	54	U	86	u
23	6	55	V	87	v
24	7	56	W	88	w
25	8	57	X	89	x
26	9	58	Y	90	y
27	:	59	Z	91	z
28	;	60	[	92	{
29	<	61	\	93	|
30	=	62	]	94	}
31	>	63	^	95	~
32	?	64	_	96	*DEL*

Wyse Attribute Code Table

Code	Attribute Type
0	Normal
1	Invisible
2	Blink
3	Invisible
4	Reverse
5	Reverse, invisible
6	Reverse, blink
7	Reverse, invisible
8	Underline
9	Underline, Invisible
:	Underline, blink
;	Underline, blink, invisible
<	Underline, reverse
=	Underline, reverse, invisible
>	Underline, reverse, blink
?	Underline, reverse, blink, invisible
p	Dim
q	Dim, invisible
r	Dim, blink
s	Dim, invisible
t	Dim, reverse
u	Dim, reverse, invisible
v	Dim, reverse, blink
w	Dim, reverse, invisible
x	Dim, underline
y	Dim, underline, invisible
z	Dim, underline, blink
{	Dim, underline, blink, invisible
|	Dim, underline, reverse
}	Dim, underline, reverse, invisible
~	Dim, underline, reverse, blink

Wyse Graphic Character Table

Graphic	Code	Graphic	Code
┬	0	┼	8
└	1	┤	9
┌	2	─	:
┐	3	▒	;
├	4	═	<
┘	5	┴	=
│	6	╫	>
▓	7	▒	?

Wyse Function Key Table

Key	Normal Value	Shifted Value	Normal Field	Shifted Field
F1	@	`	0	P
F2	A	a	1	Q
F3	B	b	2	R
F4	C	c	3	S
F5	D	d	4	T
F6	E	e	5	U
F7	F	f	6	V
F8	G	g	7	W
F9	H	h	8	X
F10	I	i	9	Y
F11	J	j	:	Z
F12	K	k	;	[
F13	L	l	<	\
F14	M	m	=	]
F15	N	n	>	^
F16	O	o	?	_
BKSP	"	'		
TAB	!	&		
INS	Q	p		
DEL	5	6		
HOME	*	/		
END	[	]		
PGUP	:	;		
PGDN	R	w		
LEFT	-	2		
RIGHT	.	3		
UP	+	0		
DOWN	,	1		
ESC	*SPACE*	%		
ENTER	$	)		
KPD ENTER	S	4		

Wyse Key Code Table

Key sequences generated when using Wyse 50 & Wyse 60 emulations

PC Key	Terminal Key	Sequence
BKSP	**BKSP**	*BS*
SHIFT+BKSP	**SHIFT+BKSP**	*BS*
TAB	**TAB**	*HT*
SHIFT+TAB	**SHIFT+TAB**	*ESC* I
INS	**INS**	*ESC* q
SHIFT+INS	**SHIFT+INS**	*ESC* r
DEL	**DEL**	*ESC* W
SHIFT+DEL	**DEL LINE**	*ESC* R
HOME	**HOME**	*RS*
SHIFT+HOME	**SHIFT+HOME**	*ESC* {
END	**CLR LINE**	*ESC* T
SHIFT+END	**CLR SCRN**	*ESC* Y
PGUP	**PGUP**	*ESC* J
SHIFT+PGUP	**SHIFT+PGUP**	*ESC* J
PGDN	**PGDN**	*ESC* K
SHIFT+PGDN	**SHIFT+PGDN**	*ESC* K
LEFT	**LEFT**	*BS*
SHIFT+LEFT	**SHIFT+LEFT**	*BS*
RIGHT	**RIGHT**	*FF*
SHIFT+RIGHT	**SHIFT+RIGHT**	*FF*
UP	**UP**	*VT*
SHIFT+UP	**SHIFT+UP**	*VT*
DOWN	**DOWN**	*LF*
SHIFT+DOWN	**SHIFT+DOWN**	*LF*
ESC	**ESC**	*ESC*
SHIFT+ESC	**SHIFT+ESC**	*ESC*
ENTER	**RETURN**	*CR*
SHIFT+ENTER	**SHIFT+ENTER**	*CR*
KPD ENTER	**ENTER**	*CR*
SHIFT+KPD ENTER	**SHIFT+ENTER**	*CR*

Wyse Key Code Table (continued)

Key sequences generated when using Wyse 50 & Wyse 60 emulations

F1	**F1**	*SOH* @ *CR*
SHIFT+F1	**SHIFT+F1**	*SOH* ` *CR*
F2	**F2**	*SOH* A *CR*
SHIFT+F2	**SHIFT+F2**	*SOH* a *CR*
F3	**F3**	*SOH* B *CR*
SHIFT+F3	**SHIFT+F3**	*SOH* b *CR*
F4	**F4**	*SOH* C *CR*
SHIFT+F4	**SHIFT+F4**	*SOH* c *CR*
F5	**F5**	*SOH* D *CR*
SHIFT+F5	**SHIFT+F5**	*SOH* d *CR*
F6	**F6**	*SOH* E *CR*
SHIFT+F6	**SHIFT+F6**	*SOH* e *CR*
F7	**F7**	*SOH* F *CR*
SHIFT+F7	**SHIFT+F7**	*SOH* f *CR*
F8	**F8**	*SOH* G *CR*
SHIFT+F8	**SHIFT+F8**	*SOH* g *CR*
F9	**F9**	*SOH* H *CR*
SHIFT+F9	**SHIFT+F9**	*SOH* h *CR*
F10	**F10**	*SOH* I *CR*
SHIFT+F10	**SHIFT+F10**	*SOH* i *CR*
F11	**F11**	*SOH* J *CR*
SHIFT+F11	**SHIFT+F11**	*SOH* j *CR*
F12	**F12**	*SOH* K *CR*
SHIFT+F12	**SHIFT+F12**	*SOH* k *CR*
CTRL+F1	**F11**	*SOH* J *CR*
SHIFT+CTRL+F1	**SHIFT+F11**	*SOH* j *CR*
CTRL+F2	**F12**	*SOH* K *CR*
SHIFT+CTRL+F2	**SHIFT+F12**	*SOH* k *CR*
CTRL+F3	**F13**	*SOH* L *CR*
SHIFT+CTRL+F3	**SHIFT+F13**	*SOH* l *CR*
CTRL+F4	**F14**	*SOH* M *CR*
SHIFT+CTRL+F4	**SHIFT+F14**	*SOH* m *CR*
CTRL+F5	**F15**	*SOH* N *CR*
SHIFT+CTRL+F5	**SHIFT+F15**	*SOH* n *CR*
CTRL+F6	**F16**	*SOH* O *CR*
SHIFT+CTRL+F6	**SHIFT+F16**	*SOH* o *CR*

APPENDIX B

Viewpoint Tables

Viewpoint Cursor Address Table

Column	Code	Column	Code	Column	Code
1	*NUL*	36	5	71	p
2	*SOH*	37	6	72	q
3	*STX*	38	7	73	r
4	*ETX*	39	8	74	s
5	*EOT*	40	9	75	t
6	*ENQ*	41	@	76	u
7	*ACK*	42	A	77	v
8	*BEL*	43	B	78	w
9	*BS*	44	C	79	x
10	*HT*	45	D	80	y
11	*DLE*	46	E		
12	*DC1*	47	F	Row	Code
13	*DC2*	48	G	1	@
14	*DC3*	49	H	2	A
15	*DC4*	50	I	3	B
16	*NAK*	51	P	4	C
17	*SYN*	52	Q	5	D
18	*ETB*	53	R	6	E
19	*CAN*	54	S	7	F
20	*EM*	55	T	8	G
21	*SPACE*	56	U	9	H
22	!	57	V	10	I
23	"	58	W	11	J
24	#	59	X	12	K
25	$	60	Y	13	L
26	%	61	\	14	M
27	&	62	a	15	N
28	'	63	b	16	O
29	(	64	c	17	P
30	)	65	d	18	Q
31	0	66	e	19	R
32	1	67	f	20	S
33	2	68	g	21	T
34	3	69	h	22	U
35	4	70	i	23	V
				24	W

Viewpoint Attribute Code Table

Code	Attribute Type
@	Normal
A	Dim
B	Normal Blinking
C	Dim Blinking
P	Reverse
Q	Dim Reverse
R	Reverse Blinking
S	Dim Reverse Blinking
`	Underlined
a	Dim Underlined
b	Underlined Blinking
c	Dim Underline Blinking
d	Invisible

Viewpoint 60 Graphic Character Table

Graphic	Code
┌	@ or A or B or C
┐	D or E or F or G
└	H or I or J or K
┘	L or M or N or O
┬	P or Q or R or S
┤	T or U or V or W
├	X or Y or Z or [
┴	\ or] or ^ or _
─	` or a or b or c
│	d or e or f or g
┼	h or i or j or k

Viewpoint Function Key Table

Key	Normal Key	Shifted Key
F1	NUL	DLE
F2	SOH	DC1
F3	STX	DC2
F4	ETX	DC3
F5	EOT	DC4
F6	ENQ	NAK
F7	ACK	SYN
F8	BEL	ETB
F9	BS	CAN
F10	HT	EM
F11	LF	SUB
F12	VT	ESC
F13	FF	FS
F14	CR	GS
F15	SO	RS
F16	SI	US
BKSP	7	W
TAB	8	X
INS	;	[
HOME	SPACE	@
END	#	C
PGUP	!	A
PGDN	"	B
LEFT	%	E
RIGHT	&	F
UP	'	G
DOWN	$	D
ESC	6	V
ENTER	9	Y
KPD ENTER	5	U

Viewpoint Key Code Table

Key sequences generated when using ADDS Viewpoint A2, Viewpoint Enhanced, Viewpoint 60 and Procomm VP60 emulations

PC Key	Terminal Key	Sequence
BKSP	**BKSP**	*BS*
SHIFT+BKSP	**SHIFT+BKSP**	*BS*
TAB	**TAB**	*HT*
SHIFT+TAB	**SHIFT+TAB**	*ESC* O
INS	**INS**	*ESC* q
SHIFT+INS	**SHIFT+INS**	*ESC* r
DEL	**DEL**	*ESC* W
SHIFT+DEL	**DEL LINE**	*ESC* l
HOME	**HOME**	*SOH*
SHIFT+HOME	**SHIFT+HOME**	*SOH*
END	**CLR LINE**	*ESC* K
SHIFT+END	**CLR SCRN**	*ESC* k
PGUP	**PGUP**	*ESC* J
SHIFT+PGUP	**SHIFT+PGUP**	*ESC* J
PGDN	**PGDN**	*ESC* |
SHIFT+PGDN	**SHIFT+PGDN**	*ESC* |
LEFT	**LEFT**	*NAK*
SHIFT+LEFT	**SHIFT+LEFT**	*NAK*
RIGHT	**RIGHT**	*ACK*
SHIFT+RIGHT	**SHIFT+RIGHT**	*ACK*
UP	**UP**	*SUB*
SHIFT+UP	**SHIFT+UP**	*SUB*
DOWN	**DOWN**	*LF*
SHIFT+DOWN	**SHIFT+DOWN**	*LF*
ESC	**ESC**	*ESC*
SHIFT+ESC	**SHIFT+ESC**	*ESC*
ENTER	**RETURN**	*CR*
SHIFT+ENTER	**SHIFT+ENTER**	*CR*
KPD ENTER	**ENTER**	*CR*
SHIFT+KPD ENTER	**SHIFT+ENTER**	*CR*

PC Key	Terminal Key	Sequence
F1	**F1**	*STX* 1 *CR*
SHIFT+F1	**SHIFT+F1**	*STX* ! *CR*
F2	**F2**	*STX* 2 *CR*
SHIFT+F2	**SHIFT+F2**	*STX* " *CR*
F3	**F3**	*STX* 3 *CR*
SHIFT+F3	**SHIFT+F3**	*STX* # *CR*
F4	**F4**	*STX* 4 *CR*
SHIFT+F4	**SHIFT+F4**	*STX* $ *CR*
F5	**F5**	*STX* 5 *CR*
SHIFT+F5	**SHIFT+F5**	*STX* % *CR*
F6	**F6**	*STX* 6 *CR*
SHIFT+F6	**SHIFT+F6**	*STX* & *CR*
F7	**F7**	*STX* 7 *CR*
SHIFT+F7	**SHIFT+F7**	*STX* ' *CR*
F8	**F8**	*STX* 8 *CR*
SHIFT+F8	**SHIFT+F8**	*STX* (*CR*
F9	**F9**	*STX* 9 *CR*
SHIFT+F9	**SHIFT+F9**	*STX*) *CR*
F10	**F10**	*STX* : *CR*
SHIFT+F10	**SHIFT+F10**	*STX* * *CR*
F11	**F11**	*STX* ; *CR*
SHIFT+F11	**SHIFT+F11**	*STX* + *CR*
F12	**F12**	*STX* < *CR*
SHIFT+F12	**SHIFT+F12**	*STX* , *CR*
CTRL+F1	**F11**	*STX* ; *CR*
SHIFT+CTRL+F1	**SHIFT+F11**	*STX* + *CR*
CTRL+F2	**F12**	*STX* < *CR*
SHIFT+CTRL+F2	**SHIFT+F12**	*STX* , *CR*
CTRL+F3	**F13**	*STX* = *CR*
SHIFT+CTRL+F3	**SHIFT+F13**	*STX* - *CR*
CTRL+F4	**F14**	*STX* > *CR*
SHIFT+CTRL+F4	**SHIFT+F14**	*STX* . *CR*
CTRL+F5	**F15**	*STX* ? *CR*
SHIFT+CTRL+F5	**SHIFT+F15**	*STX* / *CR*
CTRL+F6	**F16**	*STX* 0 *CR*
SHIFT+CTRL+F6	**SHIFT+F16**	*STX SPACE CR*

APPENDIX C

ANSI Tables

ANSI Attribute Code Table

Code	Attribute Type
0	Normal
1	Bold
2	Dim
4	Underline
5	Blinking
7	Reverse
8	Blank
22	Dim & Bold off
24	Underline off
25	Blinking off
27	Reverse off
28	Blank off
30	Black character
31	Red character
32	Green character
33	Yellow character
34	Blue character
35	Magenta character
36	Cyan character
37	White character
40	Black background
41	Red background
42	Green background
43	Yellow background
44	Blue background
45	Magenta background
46	Cyan background
47	White background

ANSI Function Key Table

Key to Program	Key
F6	17
F7	18
F8	19
F9	20
F10	21
F11	23
F12	24
F13	25
F14	26
F15	28
F16	29
F17	31
F18	32
F19	33
F20	34

ANSI Extended Key Table

Key to Program	Key
F1	112
F2	113
F3	114
F4	115
F5	116
F6	117
F7	118
F8	119
F9	120
F10	121
F11	122
F12	123
BKSP	14
TAB	16
INS	75
DEL	76
HOME	80
END	81
PGUP	85
PGDN	86
LEFT	79
RIGHT	89
UP	83
DOWN	84
ESC	110
ENTER	42
KPD ENTER	108

Modifier Keys	Mod
NONE	0
SHIFT	2
ALT	3
ALT+SHIFT	4
CTRL	5
SHIFT+CTRL	6
ALT+CTRL	7
ALT+CTRL+SHIFT	8

ANSI Key Code Table

Key sequences generated when using VT-220 or ANSI emulations

PC Key	Terminal Key	Sequence
BKSP	**BKSP**	*BS*
SHIFT+BKSP	**SHIFT+BKSP**	*BS*
TAB	**TAB**	*HT*
SHIFT+TAB	**SHIFT+TAB**	*CSI* Z
INS		*CSI* 2 ~
SHIFT+INS		*CSI* 2 ~
DEL	**DEL**	*DEL*
SHIFT+DEL	**DEL LINE**	*DEL*
HOME	**HOME**	*CSI* H
SHIFT+HOME	**SHIFT+HOME**	*CSI* H
END		*CSI* 1 ~
SHIFT+END		*CSI* 1 ~
PGUP	**PGUP**	*CSI* 5 ~
SHIFT+PGUP	**SHIFT+PGUP**	*CSI* 5 ~
PGDN	**PGDN**	*CSI* 6 ~
SHIFT+PGDN	**SHIFT+PGDN**	*CSI* 6 ~
LEFT	**LEFT**	*CSI* D
SHIFT+LEFT	**SHIFT+LEFT**	*CSI* D
RIGHT	**RIGHT**	*CSI* C
SHIFT+RIGHT	**SHIFT+RIGHT**	*CSI* C
UP	**UP**	*CSI* A
SHIFT+UP	**SHIFT+UP**	*CSI* A
DOWN	**DOWN**	*CSI* B
SHIFT+DOWN	**SHIFT+DOWN**	*CSI* B
ESC	**ESC**	*ESC*
SHIFT+ESC	**SHIFT+ESC**	*ESC*
ENTER	**RETURN**	*CR*
SHIFT+ENTER	**SHIFT+ENTER**	*CR*
KPD ENTER	**ENTER**	*CR*
SHIFT+KPD ENTER	**SHIFT+ENTER**	*CR*

When using 7 bit controls, ***CSI*** is sent as ***ESC*** **[** and ***SS3*** is sent as ***ESC*** **O**.

ANSI Key Code Table (continued)

PC Key	Terminal Key	Sequence
F1	**PF1**	***SS3*** **P**
F2	**PF2**	***SS3*** **Q**
F3	**PF3**	***SS3*** **R**
F4	**PF4**	***SS3*** **S**
F5		***CSI*** **M**
F6	**F6**	***CSI*** **17 ~**
F7	**F7**	***CSI*** **18 ~**
F8	**F8**	***CSI*** **19 ~**
F9	**F9**	***CSI*** **20 ~**
F10	**F10**	***CSI*** **21 ~**
F11	**F11**	***CSI*** **23 ~**
F12	**F12**	***CSI*** **24 ~**
CTRL+F1	**F11**	***CSI*** **23 ~**
CTRL+F2	**F12**	***CSI*** **24 ~**
CTRL+F3	**F13**	***CSI*** **25 ~**
CTRL+F4	**F14**	***CSI*** **26 ~**
CTRL+F5	**F15**	***CSI*** **28 ~**
CTRL+F6	**F16**	***CSI*** **29 ~**
CTRL+F7	**F17**	***CSI*** **31 ~**
CTRL+F8	**F18**	***CSI*** **32 ~**
CTRL+F9	**F19**	***CSI*** **33 ~**
CTRL+F10	**F20**	***CSI*** **34 ~**

When using 7 bit controls, ***CSI*** is sent as ***ESC*** **[** and ***SS3*** is sent as ***ESC*** **O**.

APPENDIX D

ASCII Codes

ASCII	Decimal	Hex	ASCII	Decimal	Hex
NUL	0	00	***SPACE***	32	20
SOH	1	01	!	33	21
STX	2	02	"	34	22
ETX	3	03	#	35	23
EOT	4	04	$	36	24
ENQ	5	05	%	37	25
ACK	6	06	&	38	26
BEL	7	07	'	39	27
BS	8	08	(	40	28
HT	9	09	)	41	29
LF	10	0A	*	42	2A
VT	11	0B	+	43	2B
FF	12	0C	,	44	2C
CR	13	0D	-	45	2D
SO	14	0E	.	46	2E
SI	15	0F	/	47	2F
DLE	16	10	0	48	30
DC1	17	11	1	49	31
DC2	18	12	2	50	32
DC3	19	13	3	51	33
DC4	20	14	4	52	34
NAK	21	15	5	53	35
SYN	22	16	6	54	36
ETB	23	17	7	55	37
CAN	24	18	8	56	38
EM	25	19	9	57	39
SUB	26	1A	:	58	3A
ESC	27	1B	;	59	3B
FS	28	1C	<	60	3C
GS	29	1D	=	61	3D
RS	30	1E	>	62	3E
US	31	1F	?	63	3F

ASCII Codes (continued)

@	64	40	`	96	60
A	65	41	a	97	61
B	66	42	b	98	62
C	67	43	c	99	63
D	68	44	d	100	64
E	69	45	e	101	65
F	70	46	f	102	66
G	71	47	g	103	67
H	72	48	h	104	68
I	73	49	i	105	69
J	74	4A	j	106	6A
K	75	4B	k	107	6B
L	76	4C	l	108	6C
M	77	4D	m	109	6D
N	78	4E	n	110	6E
O	79	4F	o	111	6F
P	80	50	p	112	70
Q	81	51	q	113	71
R	82	52	r	114	72
S	83	53	s	115	73
T	84	54	t	116	74
U	85	55	u	117	75
V	86	56	v	118	76
W	87	57	w	119	77
X	88	58	x	120	78
Y	89	59	y	121	79
Z	90	5A	z	122	7A
[	91	5B	{	123	7B
\	92	5C	|	124	7C
]	93	5D	}	125	7D
^	94	5E	~	126	7E
_	95	5F	***DEL***	127	7F

ASCII Codes (continued)

	128	80		160	A0
	129	81	¡	161	A1
	130	82	¢	162	A2
	131	83	£	163	A3
IND	132	84	¤	164	A4
NEL	133	85	¥	165	A5
SSA	134	86	¦	166	A6
ESA	135	87	§	167	A7
HTS	136	88	¨	168	A8
HTJ	137	89	©	169	A9
VTS	138	8A	ª	170	AA
PLD	139	8B	«	171	AB
PLU	140	8C	¬	172	AC
RI	141	8D		173	AD
SS2	142	8E	®	174	AE
SS3	143	8F	¯	175	AF
DCS	144	90	°	176	B0
PU1	145	91	±	177	B1
PU2	146	92	²	178	B2
STS	147	93	³	179	B3
CCH	148	94	´	180	B4
MW	149	95	µ	181	B5
SPA	150	96	¶	182	B6
EPA	151	97	·	183	B7
	152	98	¸	184	B8
	153	99	¹	185	B9
	154	9A	º	186	BA
CSI	155	9B	»	187	BB
ST	156	9C	¼	188	BC
OSC	157	9D	½	189	BD
PM	158	9E	¾	190	BE
APC	159	9F	¿	191	BF

ASCII Codes (continued)

À	192	C0	à	224	E0
Á	193	C1	á	225	E1
Â	194	C2	â	226	E2
Ã	195	C3	ã	227	E3
Ä	196	C4	ä	228	E4
Å	197	C5	å	229	E5
Æ	198	C6	æ	230	E6
Ç	199	C7	ç	231	E7
È	200	C8	è	232	E8
É	201	C9	é	233	E9
Ê	202	CA	ê	234	EA
Ë	203	CB	ë	235	EB
Ì	204	CC	ì	236	EC
Í	205	CD	í	237	ED
Î	206	CE	î	238	EE
Ï	207	CF	ï	239	EF
Ð	208	D0	ð	240	F0
Ñ	209	D1	ñ	241	F1
Ò	210	D2	ò	242	F2
Ó	211	D3	ó	243	F3
Ô	212	D4	ô	244	F4
Õ	213	D5	õ	245	F5
Ö	214	D6	ö	246	F6
×	215	D7	÷	247	F7
Ø	216	D8	ø	248	F8
Ù	217	D9	ù	249	F9
Ú	218	DA	ú	250	FA
Û	219	DB	û	251	FB
Ü	220	DC	ü	252	FC
Ý	221	DD	ý	253	FD
Þ	222	DE	þ	254	FE
ß	223	DF	ÿ	255	FF

APPENDIX E

Custom Features

Customizing the Installation Process
The AccuTerm 2K2 installation program (atw2k253a.exe, or similar name) will install user/vendor supplied files when placed in the VARFILES sub-directory of the program installation directory. Files, including directories, found in VARFILES\ATWIN will be copied into the target ATWIN program installation directory. You can install a customized version of ATWIN.INI by placing it in this directory, and if you need custom image files for GUI programs, place your custom images in VARFILES\ATWIN\IMAGES. Files found in VARFILES\HOME will be copied to the home directory selected in the installation wizard. The default home directory is the user's My Documents directory. You can install custom session (.atcf) and layout (.atly) files by placing them in this directory. Files found in VARFILES\WINDOWS will be copied to the user's WINDOWS directory, and files found in VARFILES\SYSTEM will be copied to the user's WINDOWS\SYSTEM directory.

You can use a custom ATWIN.INI file to specifiy the default values for several user-level settings. The first time a user runs AccuTerm 2K2, the settings in the ATWIN.INI file are copied to the registry for that user (HKEY_CURRENT_USER\Software\Asent\Atwin2k). The following settings (and their default values) are initialized from the ATWIN.INI file:

```
[Options]
ToolBar = True
StatusLine = True
SingleInstance = True
AutoClose = False
NoCloseWarning = False
SessionBar = True
LargeIcons = False
TrackKeyboardState = False
TitleFormat = 0
RecentListSize = 4
```

```
[Path]
Phone = ""
Capture = ""

[Xfer]
DefaultDir = ""
```

The AccuTerm 2K2 installation program can also run a vendor supplied installation program. The vendor supplied program must be named "varsetup.exe" and must reside in the same directory as the main AccuTerm setup program.

The AccuTerm 2K2 installation can be run in an automatic mode. Default values for several installation parameters may be specified in the SETUP.INI file, which must reside in the same directory as the main AccuTerm installation program. The SETUP.INI file can be usedto specify default values for several installation prompts:

```
[Defaults]
Maindir= program installation directory
Homedir= users home directory
Backup = directory to store backup files
Group= start menu group
Shortcut = "Y" or "N" to create desktop shortcut
Allusers = "Y" or "N" to create shortcut for all users
CDKEY1 = xxxxxxx      (first part of authorization key)
CDKEY2 = xxxxxxx      (second part of authorization key)
Name = xxxxxxx        (licensed user name)
Company = xxxxxxx     (licensed company name)
Location = xxxxxxx    (licensed location)
```

A sample SETUP.INI file is located in the main installation directory for AccuTerm 2K2.

If the setup program is run with the /S or /Q command line options, the installation is performed using the default values from the SETUP.INI file. The /Q option displays the installation progress; the /S option is completely silent.

Custom Icon, Title and Splash
It is possible to use a custom icon, splash screen graphic and window title with AccuTerm 2K2. To change the icon, add an **Icon** item to the **[Custom]** section of ATWIN.INI. To change the window title, add a **Title** item. To use a custom splash screen graphic, add a **Splash** item:

```
...
[Custom]
Icon=filename
Title=new window title
Splash=filename
...
```

To disable the splash screen entirely, add **Splash=no** to the **[Custom]** section.

LANPAR Emulation
AccuTerm's VT220 emulation can be modified to emulate the LANPAR Vision II terminal by adding the **Lanpar=True** item to the **[Custom]** section of ATWIN.INI. When LANPAR emulation is enabled, the differences to VT220 are:

Function Keys: *CSI* 2 ; *key* ; *bank* ; 0 u may be used to program function keys. *CSI* 0 ; *key* ; *bank* u may be used to execute a function key, and *CSI* 4 ; *key* ; *bank* u may be used to clear function key programming. ***Key*** is the function key number (1 - 12), and ***bank*** selects the shift status (1 = normal, 2 = shifted, 3 = control, 4 = control+shift, 5 = alt, 6 = alt+shift). The programming command must be followed by a string which contains the program data in the form of: ***delimiter switch route data delimiter***, where ***delimiter*** is a single ASCII character delimiter, ***switch*** is a "routing switch character", ***route*** is 0 (ignore data), 1 or 2 (program data to function key), ***data*** is a string of characters, including control characters to be programmed into the key, and ***delimiter*** is the terminating delimiter.

Message Window: AccuTerm only supports a single line host message, which is always displayed on the last line of the screen. *CSI* 2 v may be used to position the cursor in the message line, and clear the line. *CSI* 1 v may be used to position the cursor in the message line. *CSI* 0 v causes the cursor to leave the message line and return to the current screen. *CSI* 3 ; *n* v hides (*n* = 0) or displays (*n* > 0) the message line.

Screen Pages: AccuTerm supports up to 25 pages. Under LANPAR emulation, ***CSI*** **1 ; 0 ;** ***page*** **p** will change to page ***page***. ***CSI*** **1 ; 1 p** will change to the next page. ***CSI*** **1 ; 2 p** will change to the previous page.

Miscellaneous: ***ESC*** **# 7** prints the screen.

Index

AccuTerm Object Reference, 13
ADDS Programming, 117
    cursor positioning, 119
    erasing and editing, 120
    line graphics, 122
    operating modes, 117
    printer control, 123
    video attributes, 121
ADDS Viewpoint 60, 117
ADDS Viewpoint A2, 117
ADDS Viewpoint Enhanced, 97
ADDS Viewpoint Tables, 307
ANSI Attribute Code Table, 315
ANSI Extended Key Table, 317
ANSI Function Key Table, 316
ANSI Key Code Table, 318
ANSI Programming, 125
    character set selection, 132
    cursor positioning, 135
    erasing and editing, 138
    function key programming, 145
    operating modes, 127
    printer control, 141
    terminal reports, 142
    video attributes, 140
ANSI Tables, 315
application object, 13
ASCII Codes, 321
bitmap files, 92
CapsLock, 90
commands
    private, 85
Custom Icon, Title and Splash, 327
Custom Mouse Action, 297
Customizing the Menu and Toolbar, 77
data capture
    starting, 87
    stopping, 88
Download
    host control of, 86, 87
execute DOS command, 85
extended mode, 90
file transfer
    host control of, 87
    status of, 87
file transfer protocol
    ASCII, 86
    Kermit, 86
    overwrite, 86
    resuming Zmodem file transfer, 86
    Xmodem, 86
    Ymodem, 86
    Zmodem, 86
FTSERVER, 155
FTSETUP, 155
function keys
    programming, 91
GED verb, 166
GIF files, 92
GUI
    Code Editor, 167
    code template, 167
    Code Templates, 285
    Custom Code Templates, 285
    dialog box, 167
    Error Handling, 281
    event decoder, 168
    event driven model, 165
    Global Objects, 278
        Printer, 279
        Root, 278
        Screen, 278
    Internationalization, 286
    Multi-Application MDI Subroutine, 168

Multi-Application Subroutine, 168
Multiple Application Model, 283
properties dialog, 166
standalone, 167
standard properties, 276
tab order, 166
Tab Order, 282
Update Code, 167, 168
GUI Designer, 165
GUI Development Environment, 165
GUI Library, 165, 171
ATGUIACTIVATE subroutine, 239
ATGUIBEGINMACRO subroutine, 264
ATGUICHECKEVENT subroutine, 262
ATGUICLEAR subroutine, 256
ATGUICREATEAPP subroutine, 178
ATGUICREATEBUTTON subroutine, 206
ATGUICREATECHECK subroutine, 203
ATGUICREATECOMBO subroutine, 195
ATGUICREATEEDIT subroutine, 186
ATGUICREATEFORM subroutine, 183
ATGUICREATEFRAME subroutine, 220
ATGUICREATEGAUGE subroutine, 230
ATGUICREATEGRID subroutine, 209
ATGUICREATELABEL subroutine, 189
ATGUICREATELIST subroutine, 191
ATGUICREATEMENU subroutine, 232
ATGUICREATEOPTION subroutine, 199
ATGUICREATEPICTURE subroutine, 218
ATGUICREATETAB subroutine, 224
ATGUICREATETABGRP subroutine, 222
ATGUICREATETOOLBAR subroutine, 235
ATGUICREATETREE subroutine, 226
ATGUIDELETE subroutine, 238
ATGUIDISABLE subroutine, 243
ATGUIENABLE subroutine, 242
ATGUIENDMACRO subroutine, 264
ATGUIFILEDIALOG subroutine, 273
ATGUIFONTDIALOG subroutine, 275
ATGUIGETACTIVE subroutine, 245
ATGUIGETITEMPROP subroutine, 251
ATGUIGETPROP subroutine, 247
ATGUIGETPROPS subroutine, 249
ATGUIGETUPDATES subroutine, 254
ATGUIGETVALUES subroutine, 253
ATGUIHELP subroutine, 268

ATGUIHIDE subroutine, 241
ATGUIINIT2 subroutine, 175
ATGUIINPUTBOX subroutine, 272
ATGUIINSERT subroutine, 257
ATGUIINSERTITEMS subroutine, 258
ATGUILOADVALUES subroutine, 252
ATGUIMOVE subroutine, 244
ATGUIMSGBOX subroutine, 270
ATGUIPOSTEVENT subroutine, 263
ATGUIPRINT2 subroutine, 267
ATGUIREMOVE subroutine, 259
ATGUIREMOVEITEMS subroutine, 260
ATGUIRESET subroutine, 255
ATGUIRESUME subroutine, 177
ATGUIRUNMACRO subroutine, 264, 266
ATGUISETITEMPROP subroutine, 250
ATGUISETPROP subroutine, 246
ATGUISETPROPS subroutine, 248
ATGUISHOW subroutine, 240
ATGUISHUTDOWN subroutine, 176
ATGUISUSPEND subroutine, 177
ATGUIWAITEVENT subroutine, 261
GUI project, 166
GUI Runtime, 170
image display
host control, 92
JPEG files, 92
LANPAR Emulation, 327
license
determining license type, 88
macro commands
host control, 91
metafiles, 92
MIDI sound files, 90
mouse
host control of, 85
Mouse Pattern Table, 297
MultiValue Server Object, 155
network
Modular Software PicLan, 325
PicLan, 325
TCP/IP, 325
Telnet, 325
normal mode, 90
Object Bridge, 289
object hierarchy, 13
Object Reference, 13
AccuTerm object, 15
Creating new session, 24
Events, 14
Menu object, 72
Methods, 14
MnuBand object, 73
MnuTool object, 74
Predefined session objects, 24
Properties, 14
ScreenBlock object, 71
Session object, 23, 24
Sessions collection, 23
Settings object, 57
Type Library, 14

Optional File Installation, 325
Pick PC Console Programming, 149
cursor positioning, 149
erasing and editing, 150
operating modes, 149
printer control, 153
protect mode, 152
video attributes, 151
Private Commands, 85
Procomm VP60, 117
programming
AccuTerm, 85
release, 88
Script
AppActivate statement, 8
AppClose statement, 8
AppFind function, 8
AppGetActive function, 8
AppGetPosition statement, 9
AppGetState function, 9
AppHide statement, 9
AppList statement, 9
AppMaximize statement, 9
AppMinimize statement, 9
AppMove statement, 9
AppRestore statement, 9
AppSetState statement, 10
AppShow statement, 10
AppSize statement, 10
Chain statement, 10
closing, 4
controlling AccuTerm, 7
creating, 3
debugging, 6
Debugging, 11
editing, 5
FileExists function, 11
Item function, 11
ItemCount function, 11
Language Extensions, 8
Line function, 11
LineCount function, 11
loading, 4
OpenFileName function, 11
Pause statement, 12
printing, 4
Random function, 12
running, 5
running from command line, 5
running from function key, 6
running from host computer, 6
running from menu or toolbar, 6
saved in layout file, 5
SaveFileName function, 12
saving, 4
Word function, 12
WordCount function, 12
Scripting, 3, 13
serial number, 88
Server Functions, 158
Server Properties, 162
sounds
host control, 90
TARGA files, 92
TIFF files, 92
Upload
host control of, 87
Using the Menu Designer, 77
VBA, 3
Viewpoint 60, 117
Viewpoint A2, 117
Viewpoint Attribute Code Table, 309
Viewpoint Cursor Address Table, 307
Viewpoint Function Key Table, 310
Viewpoint Key Code Table, 311
Viewpoint Tables, 307
Visual Basic for Applications, 3
VT-100 Programming, 125
VT-220 Programming, 125

VT-320 Programming, 125
VT-420 Programming, 125
VT-52 Programming, 147
Wave sound files, 90
wED editor, 167
Wyse 50, 97
Wyse 60, 97
Wyse Attribute Code Table, 302
Wyse Cursor Address Table, 301
Wyse Function Key Table, 304
Wyse Graphic Character Table, 303
Wyse Key Code Table, 305
Wyse Programming, 97
  attributes, 108
  character set selection, 102
  cursor positioning, 103
  erasing and editing, 106
  function key programming, 114
  line graphics, 111
  operating modes, 97
  printer control, 112
  protect mode, 110
Wyse Tables, 301