Revision 4.0 **December 14, 2000**

AccuTerm 2000

Programmers Guide

MultiValue Edition

ACCUSOFT ENTERPRISES **WWW.ASENT.COM**
8041 Foothill Blvd. **Phone (818) 951-1891**
Sunland, California 91040 **FAX (818) 951-3606**

Contents

INTRODUCTION

This Programming Reference manual is designed to provide developers (and power users) technical information about AccuTerm 2000. Information on the programming codes for all of the terminals which are emulated by AccuTerm 2000 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 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, 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** option.

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 global variables, 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 `Private`, `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.

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 command line, append the name of the script file to the shortcut target or command line (to modify the shortcut target field, click the AccuTerm icon with the *right* mouse button, then select **Properties**, then click the **Shortcut** tab). 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.

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. Usually, a main script would be loaded in the script window, and the programmed key would simply call one of the subroutines in the main script (possibly with arguments), or the programmed key would "chain" to another script file.

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** application, set the **Action** property of the menu or toolbar item to the desired script statements enclosed in brackets [ ].

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 carriage return. Often, when a main script is loaded by opening a layout file (from the command line), the script command sent by the host is simply the name of a subroutine or macro to call (possibly with arguments), or a **Chain** statement which executes another script file.

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 which can be manipulated. The main object, `AccuTerm`, may be used to control the general settings of AccuTerm - 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 the object structure used to control AccuTerm.

There are several built-in objects which can be referenced while executing scripts. 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 `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 may not be the same as `ActiveSession`, since script execution could be initiated from a non-active session under host program control.

Script Language Extensions
Several new statements 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 e 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.SCR|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 available file types.

`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.

`Randon` 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 available file types.

`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***.

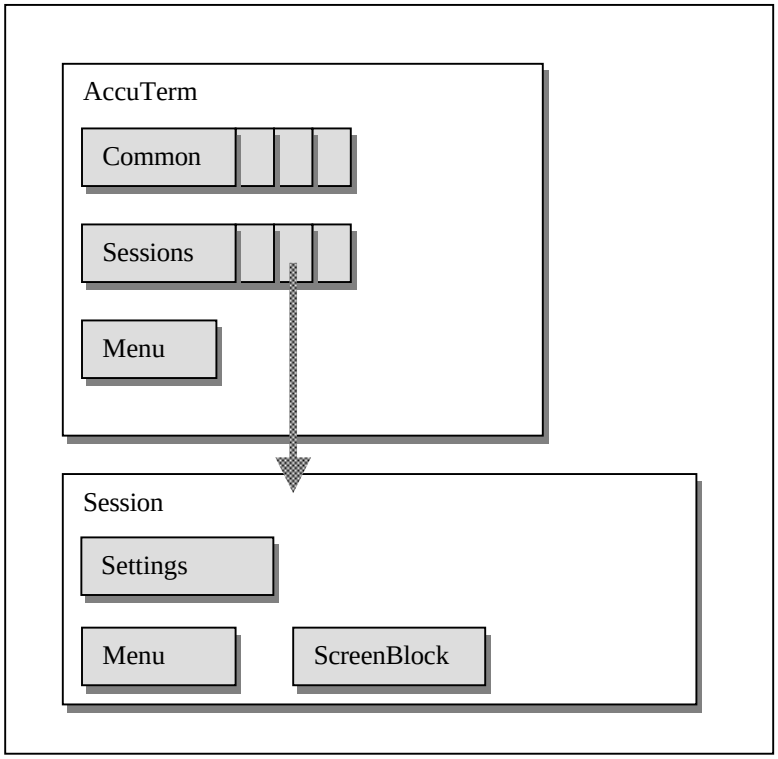
`WordCount` function

`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 2000 exposes a rich ActiveX object structure. 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 2000 includes a type library, ATWIN2K.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", 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.

`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.

`Height` property (long)
This is the application window height in pixels.

`Hide` method
This method makes AccuTerm's main window invisible.

`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".

`MenuFileName` property (string)
This is the name of a custom menu file. Use the menu designer to create custom menus. For more information on the menu designer, refer to the 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.

`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 license, 3=enterprise license, 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 2000.

`ProductSerialNumber` property (long)
This read-only property returns the serial number of AccuTerm 2000.

`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.

`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)`.

`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.

`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.

`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 "sessions" to communicate with host computer systems. 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 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 (create, 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, the host computer system, or from DDE, 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.

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

`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.

`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. If ***source*** is zero or ***mode*** is 3, 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). 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.

`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
Blue	1	16
Green	2	32
Cyan	3	48
Red	4	64
Magenta	5	80
Brown	6	96
Light Grey	7	112
Dark Grey	8	128
Light Blue	9	144
Light Green	10	160
Light Cyan	11	176
Light Red	12	192
Light 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

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

`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.

`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.

`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
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.

`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. 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

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		

`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*** `[,` ***NoProtect*** `]]]])`

Returns the text contents of a row on the session screen. Default coordinate is the current cursor location. Default number of columns is to the end of the row. If ***NoProtect*** is True (non-zero), only un-protected characters are returned (protected characters are returned as blanks), otherwise, all ANSI characters are returned.

`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 “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.

`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.

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

`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.

`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.

`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.

`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, 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.

`LockHistory` property (boolean)

Set to True (non-zero) to prevent the history buffer from being updated; otherwise set to False (zero).

`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".

`MenuFileName` property (string)
This is the name of a custom menu file. Use the menu designer to create custom menus. For more information on the menu designer, refer to the chapter titled "Customizing the Menu and Toolbar".

`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.

`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` or `atDevMODEM`. Acceptable values are from 1 to 8. 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 in non-zero (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 in non-zero (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 in non-zero (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 in non-zero (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 auxiliary port output. It may be set to null ("") to indicate “no printer”.

`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” 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 in non-zero (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 in non-zero (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

`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 in non-zero (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.

`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.

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

`ScrAutoFwd` property (boolean)
Set to True (non-zero) to enable the "Screen Auto Forward" feature; 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.

`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.

`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 (boolean)
Set to False (zero) for text mode, True (non-zero) for graphics mode. When this property is True, the PrinterFont…, PrinterOrientation, PrinterPaperSize and PrinterPaperSource properties affect how the slave (Aux) print job is printed.

`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).

`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. *Only cipher 0 (3DES) is valid at this time*.

`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.)

`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.

`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) 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. This option is only available when the u2licn.dll file is installed on your computer. The u2licn.dll file is not available from AccuSoft. Contact your Informix/Ardent 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.

`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
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.

`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 properties as a group. When changing multiple properties, using the `Settings` object 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.

The recommended way to use the Settings object is in a `With Session.Settings . . . End With` construct.

`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.

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

`AttributeMask` property (integer)
The AttributeMask property indicates which visual effects are to be displayed. Use 10 for blink and underline, 8 for underline, 2 for blink or zero to disable both blinking and underline.

`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.

`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 = function keys
M = miscellaneous

`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 (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.

`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`**.

`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` or `atDevMODEM`. Acceptable values are from 1 to 8. 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 in non-zero (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 in non-zero (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 in non-zero (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 in non-zero (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 auxiliary port output. It may be set to null ("") to indicate “no printer”.

`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 in non-zero (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 in non-zero (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 in non-zero (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).

`ScrAutoFwd` property (boolean)
Set to True (non-zero) to enable the “Screen Auto Forward” feature; 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 (boolean)
Set to False (zero) for text mode, True (non-zero) for graphics mode. When this property is True, the PrinterFont…, PrinterOrientation, PrinterPaperSize and PrinterPaperSource properties affect how the slave (Aux) print job is printed.

`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).

`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. *Only cipher 0 (3DES) is valid at this time*.

`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.)

`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.

`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) 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. This option is only available when the u2licn.dll file is installed on your computer. The u2licn.dll file is not available from AccuSoft. Contact your Informix/Ardent 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 (integer)
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
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 member of the AccuTerm object is the operational menu. When any session is open, the menu object member of the active session is the operational menu. The menu object contains a number of `MnuBand` objects, which contain either `MnuTool` objects or sub-menus (other `MnuBand` objects).

`AddNew` method (`MnuBand` object)

```
Set MnuBandObject = MenuObject.AddNew(name)
```

This method creates a new MnuBand object in the specified Menu object. After creating a new MnuBand object, use the MnuBand.AddNew method to create MnuTool objects for the newly created band.

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

MnuBands property (Collection of MnuBand objects)
This read-only property returns the collection of MnuBand objects contained in the menu.

RecalcLayout method
This method recalculates the menu band. Use this method after modifying properties of any contained bands or tools.

Remove method

MenuObject*.Remove *name

This method removes the specified band from the menu.

The MnuBand Object

A Menu object contains zero or more MnuBand objects. Examples of MnuBand objects are the main menu, the toolbar, the status bar, popup menus and sub-menus of the main menu or a popup menu. A MnuBand object contains either MnuTool objects or sub-menus (other MnuBand objects).

AddNew method (MnuTool object)

Set ***MnuToolObject*** = ***MnuBandObject*.AddNew(*ID*)**

This method creates a new MnuTool object with tool ID ***ID*** in the specified MnuBand object.

BandType property (integer)
This property is the type of band: 0 = toolbar or status bar, 1 = main menu, 2 = popup menu or sub-menu.

Caption property (string)

This property is used for sub-menus, and is the caption displayed in the higher-level menu item.

`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.

`Insert` method
MnuBandObject*`.Insert` *MnuToolObject, AfterID
This method inserts a `MnuTool` object at a specified location in the band. ***AfterID*** is the tool ID to insert the tool after, and is a string.

`MnuTools` property (Collection of `MnuTool` objects)
This read-only property returns the collection of `MnuTool` objects contained in the menu band.

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

`Remove` method
MnuBandObject*`.Remove` *ID
This method removes a `MnuTool` object from the band. ***ID*** is the tool ID to remove, and is a string.

`Visible` property (string)
This property is True 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, icon, 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, but an optional action can be specified by setting this property. Set this property to null to perform the default action. Otherwise, to execute a macro, enclose the macro statement in square brackets (`[ ]`). To transmit a character sequence to the host, set this property to the desired sequence.

`BeginGroup` property (integer)
Set this property to True to place a separator between this tool and the one before it.

`Caption` property (string)
This property is the caption displayed in the higher-level menu item. The caption is not displayed in a toolbar.

`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.

`Height` property (long)
To set the height of the menu item to a specific size, set this property to the desired height in pixels. If this property is zero, the default height is used.

`ImageNo` property (integer)
Set this property to the ID of the icon to display in the menu or toolbar of the menu tool.

`Style` property (integer)
This property is type of tool: 0 is normal (icon only if a toolbar, otherwise icon and caption), 1 is caption only, 2 is icon only, and 3 is both caption and icon.

`SubBand` property (string)
This property is the name of the `MnuBand` object to be opened as a sub-menu when the tool is clicked.

`ToolID` property (long)

This property is the tool ID.

`ToolTipText` property (string)
This property is the text which is displayed in a floating window when the cursor is paused over a menu item or toolbar button.

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

`Width` property (long)
To set the width of the menu item to a specific size, set this property to the desired width in pixels. If this property is zero, the default width is used.

CUSTOMIZING THE MENU AND TOOLBAR

AccuTerm 2000 includes a menu design utility which can be used to customize AccuTerm's main menu, popup 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.

Menu Structure

The menu file stores the complete menu structure used by AccuTerm. The menu structure is hierarchical, and consists of zero or more top-level menu items: Menu Bar, ToolBar, Status Line, and Popup Menu. You can remove any of these items, or add new ones as desired (it is possible to have more than one main Menu Bar, ToolBar, Status Line or Popup Menu). Each top-level menu item contains "tools". Each "tool" can specify an action, or open a sub-menu. There are built-in menu tools, one for each of the menu or toolbar actions included on AccuTerm's default menus. You can modify the built-in tools, but they cannot be deleted from the tool library. You can create your own tools as well.

Main Menu

The main menu is displayed when AccuTerm is first started, or when all session windows have been closed. This menu is unique in that no session-specific items are included. As soon as a session is opened, the main menu is replaced by the session menu. AccuTerm 2000 loads the main menu from the **dfltmain.mnu** file. If you want to modify the main menu, you should modify this file. If you need to restore the main menu to its original state, simply copy the **stdmain.mnu** file over **dfltmain.mnu**.

Session Menu

Once a session is open, AccuTerm replaces the main menu with the session menu. By default, AccuTerm loads the session menu from the **dfltsess.mnu** file. This is the menu which is displayed when a new session is created. Optionally, a **.CFG** file can specify an alternate menu to use when the session is opened. Use **File**

Update Session from the menu designer to modify the **.CFG** file with an alternate session menu. If you need to restore the default session menu to its original state, simply copy the **stdsess.mnu** file over **dfltsess.mnu**.

Menu Bar

The Menu Bar is the standard Windows menu, displayed at the top of AccuTerm's main window. Normally, items on this menu open up sub-menus; for example, clicking on the Menu Bar **File** menu item opens up the File sub-menu, which contains tools for New, Open, Save, etc. Properties of the Menu Bar are Name, Caption, Visible and DockingArea.

Name: name used to reference this menu item.

Caption: displayed only if the Menu Bar is "floating".

Visible: check this box if the Menu Bar is visible.

DockingArea: 1 if Menu Bar is "docked" to the top of the window (default), 2 if bottom, 4 if left side, 8 if right side.

ToolBar or StatusLine

The ToolBar is the button bar displayed just below the Menu Bar at the top of AccuTerm's main window. Normally, items on the ToolBar are "tools" with icons, but without captions, which perform some action when clicked. The default tool bar has buttons for New, Open, Save, etc. You can create your own tools, and add them to the ToolBar.

The Status Line displays informaton about the state of AccuTerm at the bottom of AccuTerm's main window. Normally, items on the Status Line are "tools" with icons and caption, but which do not perform any action when clicked. The default status line has tools for the current cursor position, screen size, emulation, etc.

Properties of the ToolBar / Status Line are Name, Caption, Visible, and DockingArea.

Name: name used to reference this menu item.

Caption: displayed only if the ToolBar or Status Line is "floating".

Visible: check this box if the item is visible.

DockingArea: 1 if the item is "docked" to the top of the window (default for ToolBar items), 2 if "docked" to the bottom of the window (default for Status Line items), 4 if left side, 8 if right side.

Popup Menu

The Popup Menu (context menu) is displayed when the user clicks the AccuTerm screen with the right mouse button. Normally, items on this menu perform some action, like Copy, Paste, etc. You can use the **Custom Mouse Action** (described earlier in this section) to open your own popup menu when the mouse is clicked over a specified character pattern. Properties of the Popup Menu are Name, Caption and Visible.

Name: name used to reference this Popup Menu.

Caption: not used in Popup Menus.

Visible: check this box if the item is visible.

Sub-Menu

A sub-menu can be attached to any of the top-level menu items (Menu Bar, ToolBar, Status Line, Popup Menu). A sub-menu is opened when the user clicks its caption or button on its parent menu. The sub-menu can contain other sub-menus, or "tools" which perform some action. Properties of the Sub-Menu are Name, Caption, Visible and BeginGroup.

Name: name used to reference this menu item.

Caption: text is displayed on the menu or toolbar. For menu items, one letter of the caption may be marked as a "hot key" by preceding that letter with an ampersand (&).

Visible: check this box if the sub-menu is visible.

Begin Group: check this box to place a separator between this item and the tool preceding it in the parent menu.

Tool

A “tool” is a menu item which performs some action. A tool has a Caption, Tool Tip Text, Action, and optional Icon. A tool which performs an action acts like a button (the Button check-box is checked). Tools on the Status Line usually do not perform any action, so they do not have the Button check-box checked. Other tool properties are Height, Width, Begin Group, Visible, and Enabled.

Caption: text is displayed on the menu or toolbar. For menu items, one letter of the caption may be marked as a “hot key” by preceding that letter with an ampersand (&).

Tool Tip Text: text displayed in a floating window when the mouse is paused over the menu item.

Action: macro name to execute enclosed in square brackets ([ ]) or character sequence to transmit to the host. Control characters are entered as described in “function key programming”. If this field is blank, the default action for the built-in tool is performed.

Height: fixed height in pixels. Leave this field blank for auto-height.

Width: fixed width in pixels. Leave this field blank for auto-width.

Icon: the icon displayed in the menu, toolbar or status line. Click Select Icon to select a new icon; click Clear Icon to remove the icon from the tool.

Begin Group: check this box to place a separator between this tool and the tool preceding it in the parent menu.

Visible: check this box if the tool is to appear in the menu, toolbar or status line. For built-in tools, this property may change to reflect AccuTerm’s current state.

Enabled: check this box if the tool is to be “enabled”. If the tool is not enabled, it is displayed in a light-grey color, and no action is performed when the tool is clicked. For built-in tools, this property may change to reflect AccuTerm’s current state.

Checked: check this box to indicate that the tool is “checked”. When the tool is checked, a check-mark will be displayed next to the tool caption instead of an icon. For built-in tools, this property may change to reflect AccuTerm’s current state.

Using the Menu Designer

To start the menu designer, locate the **atmnudsg.exe** file (in \Program Files\Atwin) using Windows Explorer, and double click on the file name. Use **File Open** to open an existing menu file, then **File Save As** to save the menu under a new name.

After a menu file is opened in the menu designer, two panels are displayed: the Menu Preview, and the Menu Designer. The Menu Preview panel displays a sample screen showing the current menu file. You can exercise the menu system using this panel. The Menu Designer panel is used to manipulate the menu structure.

The menu structure is displayed as a hierarchy, with AccuTerm as the root. The top-level menu items Menu Bar, ToolBar, Status Line and Popup Menu are displayed at the next level. Under the top-level menu items are sub-menus or tools. Sub-menus may contain other sub-menus or tools.

To expand the menu display, click on the plus sign next to the item you want expanded. To collapse the display, click on the minus sign next to the item. To insert a new item into the structure, click the location where you want the new item inserted with the right mouse button, and select Add. To remove an item, click the item with the right mouse button, and select Remove. To modify an item, click the item with the right mouse button and select Properties.

When you are finished modifying the menu structure, use **File Save** to save the menu. To update a session configuration file, use **File Update Session**.

ACCUTERM PROGRAMMING

AccuTerm 2000 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 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 DOS command ***command***, then returns to AccuTerm emulation mode immediately. Command is executed concurrently with the terminal session.

ESC STX > ***command*** ***CR***

Executes the DOS command ***command*** and suspends the terminal session until the command completes.

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).

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` (); 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

ESC STX C ***o p*** ; ***path CR***

Begin capture. Mode ***o*** may be O (overwrite), A (append), N (new file only) or C (clipboard). Source ***p*** may be P to capture printed data or null to capture received 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:

C = capture supported
D = download supported
U = upload supported
A = ASCII protocol supported
K = Kermit protocol supported
I = Image display supported

E = Execute command supported
S = Scripting supported
H = Server mode supported
P = Packetized messages supported
R = Reliable connection
X = File conversion supported
T = File transfer error info supported
O = Object Bridge supported
G = GUI supported

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

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 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 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 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). Key code ***k*** may be digits 0 to 9 or : 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 macro command, 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 `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 `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	Blue
2	Green
3	Cyan
4	Red
5	Magenta
6	Brown
7	Light Gray
8	Dark Grey
9	Light Blue
10	Light Green
11	Light Cyan
12	Light Red
13	Light 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.

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.‡

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

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 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 status 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 c D Selects the primary character set.[§]

ESC c E Selects the secondary character set.[§]

[‡] Only valid in Viewpoint Enhanced mode.
[§] Only valid in Wyse 60 mode.

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 ~ ***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 ~ ; or ***ESC*** ~ < or ***ESC*** ~ = or ***ESC*** ~ A or ***ESC*** ~ B
Select VT-220 emulation.

ESC ~ > Select Tektronix 4014 emulation.

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.‡

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

---

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

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.

ESC X ***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.

‡ Only valid in Viewpoint Enhanced mode.

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

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

ESC] Selects page 0.

ESC } Selects page 1.

ESC - ***n r c***
Selects page ***n*** (pages numbered from 0 to 6) 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).

ESC w ***n*** Selects page ***n***. Pages are numbered from 0 to 6.§

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

ESC w P 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 to 6) 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).§

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

† Only valid in Wyse 50 and Wyse 60 modes.
§ Only valid in Wyse 60 mode.

(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.[†]

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).

---

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

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.

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. This command is used to start a specific attribute on the main screen. In Wyse 50 and ADDS VP mode, this command changes all characters from the current position until the end of the screen or until another attribute code character 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).

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*.

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.

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).

‡ Only valid in Viewpoint Enhanced mode.

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 carriage return.

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 6 Sends the current row to the host computer followed by a carriage return.†

ESC 7 Sends the entire screen to the host computer. Each row of the screen is followed by a carriage return.

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.

ESC b Sends the current cursor position in the 132 column format.

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".

† Not valid in ADDS Viewpoint A2 mode.

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.
§ Only valid in Wyse 60 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 c U Clear all redefinable key definitions to their default values.[§]

ESC Z ~ ***k*** Read programmable key definition for key ***k***.

[§] Only valid in Wyse 60 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 [S].

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 ofl 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.[234]

CSI ? 66 l
Cursor keys return cursor codes.[234]

CSI ? 67 h
Backspace keys sends ***BS*** control code.[34]

CSI ? 67 l
Backspace keys sends ***DEL*** control code.[34]

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.[34]

CSI ? 95 l
Clear screen when column mode changes.[34]

ESC % ! 0
Enters Tektronix 4100 graphics mode.

CSI 0 ***SP*** q or ***CSI*** 1 ***SP*** q or ***CSI*** 2 ***SP*** q
Select block cursor. [23L]

CSI 3 ***SP*** q or ***CSI*** 4 ***SP*** q
Select underscore cursor. [23L]

CSI ***n*** $ |
Set number of columns to ***n***. [3L]

CSI ***n*** t or ***CSI*** ***n*** * |
Set number of rows to ***n***. [3L]

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. [3L]

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 International.

ESC) < Sets character set G1 to International.

ESC * < Sets character set G2 to International (default).[234L]

ESC + < Sets character set G3 to International.[234L]

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 G2 to ISO-Latin1.[L]

ESC + A Sets character set G3 to ISO-Latin1.[L]

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.[234L]

ESC + 0 Sets character set G3 to graphics.[234L]

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 ISO-Latin1.[1234]

ESC) % 5 Sets character set G1 to ISO-Latin1.[1234]

ESC * % 5 Sets character set G2 to ISO-Latin1.[234]

ESC + % 5 Sets character set G3 to ISO-Latin1.[234]

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. [s]

CSI ***n*** `= G` Sets the current normal background color to ***n***. [s]

CSI ***n*** `= H` Sets the current reverse foreground color to ***n***. [s]

CSI ***n*** `= I` Sets the current reverse background color to ***n***. [s]

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.

ESC `] R`
Reset Linux Console palette.

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.[1234]

CSI `10 i` Prints the screen display ignoring the print extent in effect.[34]

CSI `? 1 i` Prints the current cursor line.[1234]

CSI `5 i` Transparent print mode. Characters are printed, but not displayed.[1234]

CSI `? 5 i` Auto print mode. 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.[1234]

CSI `4 i` or ***CSI*** `? 4 i`
Cancel transparent or auto print mode.[1234]

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:[234]
CSI ? 2 7 ; 0 ; 0 ; 0 n

CSI ? 5 5 n
Request locator status. If the mouse is currently enabled, AccuTerm will respond:[234]
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:[234]
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:[234]
CSI 0 * {

CSI ? 6 3 ; ***id*** n
Request memory checksum. AccuTerm will respond:[234]
DCS ***id*** ! ~ 0 0 0 0 ***ST***

CSI ? 7 5 n
Request data integrity. AccuTerm will respond:[234]
CSI ? 7 0 n

CSI " v
Request displayed extent. AccuTerm will respond:[34]
CSI ***rows*** ; ***columns*** ; 1 ; 1 ; ***page*** " w

CSI ***n*** $ p
Request state of ANSI mode ***n***. AccuTerm will respond:[34]
CSI ***n*** ; ***value*** $ y

CSI ? ***n*** $ p
Request state of private mode ***n***. AccuTerm will respond:[34]
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 alternate method to program function keys:

DCS `" x |` ***key / mod / fn / value*** `/ 1 ; . . .` ***ST***

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 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**, included with AccuTerm, 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 Users Guide for instructions on installing the host programs), use the **FTSETUP** utility to configure the server. You must use **FTSETUP** before you can use the server because the server name needs to be established. 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 next section.

When you are finished using the server, you can disconnect and destroy 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` method

`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` method

```
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. When establishing a connection, the ServerConnect function of the FTSERVER.SCR script is called. You can customize this script.

`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. When terminating the connection, the ServerDisconnect function of the FTSERVER.SCR script is called. You can customize this script.

`Download` method

```
Server.Download SourceFile, SourceIDs, TargetFolder
  [, Protocol [, Binary [, Overwrite ]]]
```

Uses the AccuTerm FT program to download files from the Pick 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` method

```
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 FTD program to download a file from the Pick 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).

`IConv` method

```
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 FTD program to upload a file from the PC to the Pick 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).

`OConv` method

```
Result = Server.OConv(data, code)
```

Performs an "output conversion" on ***data***. Conversion to be performed is ***code***.

`ReadItem` method

```
Result = Server.ReadItem(file, ID [, attr [, value [,
  subvalue [, locked ]]]] )
```

Reads an item from the Pick 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 Pick/BASIC READU statement). Error 260 is returned if the item was already locked by another process.

`Readnext` method

```
Result = Server.Readnext(file, attr [, value [,
  subvalue ]] )
```

Reads the next item from the Pick 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 FT program to upload files from the PC to the 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 Pick database. If the optional ***KeepLocked*** parameter is non-zero, the item is left locked after the writing (same as Pick/BASIC WRITEU statement).

GUI DEVELOPMENT ENVIRONMENT

AccuTerm 2000 GUI (Graphical User Interface) consists of four components: the AccuTerm GUI Library (Pick/BASIC subroutines), the transport mechanism (AccuTerm 2000), the GUI runtime and the GUI designer.

The GUI Designer

The GUI designer is used to create and maintain GUI application "templates". The template 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 many properties. The template is in the format required by the GUI Library ATGUIRUNMACRO subroutine. Header information is included in the template to support caching of the template 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 Pick **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 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 designer is integrated with the **wED** code editor. You can use the **wED** editor to edit the Pick/BASIC code corresponding to the GUI project you are working on. Click the Tools menu for code editing options, or select Edit Code from the popup menu displayed when right-clicking a control in the form or project tree.

The first time you access the code tools without an associated program file and item, you will need to enter a program file and item-id to be associated with your GUI project. Alternatively, the GUI designer can generate a skeleton program for you if you have not yet written one yourself. The code generator uses a template, which you can select (View Options menu). You can also create your own template files.

The GUI runtime environment supports multiple GUI applications being loaded together. In order to support this, one master program calls subroutines written to handle each GUI application. When you generate code for an application for the first time, you need to select “standalone” or “subroutine”. If you select “standalone”, a self-contained Pick/BASIC program for the GUI application will be generated. If you select “subroutine”, a callable subroutine will be generated instead. The subroutine must be called by a “main” program patterned after the included ATGUI.MAIN.SAMPLE program.

When you are finished creating or modifying a GUI project, select Save from the File menu to save the updated GUI project template back on your host system.

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* their default values. It also includes a *Changed* property which indicates if any contained *controls'* value has been modified.

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 or picture), 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 Pick/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 file transfer programs. If you need to install this at a later time, type `RUN FTBP LOAD-ACCUTERM-GUIBP` at TCL.

The structure of a typical AccuTerm GUI program is:

```
Perform application initialization
Initialize the GUI environment (ATGUIINIT)
Load template for GUI 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.

AccuTerm GUI Library Subroutines

ATGUIINIT

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

Calling syntax:

CALL ATGUIINIT(ERRORS, GUISTATE)

Output arguments:

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

Other arguments: for internal use - do not modify

GUISTATE

If an error occurs, ERRORS<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.

Calling syntax:

```
CALL ATGUISHUTDOWN
```

Output arguments: none

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, HELPFILE, RTNMODE, SCALE, LEFT,
        TOP, WIDTH, HEIGHT, ERRORS, 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	MDI application caption (ignored for SDI applications)
HELPFILE	filename for Windows help file for application
RTNMODE	non-zero if return key acts like tab key
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:

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

Other arguments: for internal use - do not modify

GUISTATE

Properties supported by this item:

GPBACKCOLOR	background color (default color is ApplicationWorkspace)
GPENABLED	non-zero if form is enabled
GPVISIBLE	non-zero if form is visible
GPPICTURE	filename of an image to be displayed as the form's background
GPCAPTION	MDI application caption (ignored for SDI applications)
GPHELPFILE	filename for Windows help file for application
GPRTNMODE	non-zero if return key acts like tab key
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
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

Events supported by this item:

GEQUIT	the GUI application has terminated – end of event processing

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.

Calling syntax:

```
CALL ATGUICREATEFORM(APPID, FORMID,
        EVENTMASK, CAPTION, TYPE, LEFT, TOP,
        WIDTH, HEIGHT, ERRORS, 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 = dialog box
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:

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

Other arguments: for internal use - do not modify

GUISTATE

Properties supported by this item:

GPFORECOLOR	foreground (text) color (default color is 3DFace)
GPBACKCOLOR	background color (default color is WindowText)
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 in the application help file
GPPICTURE	filename or URL of an image to be displayed as the form's background
GPCAPTION	form caption

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
GEDEACTIVATE	the form is no longer the active form

ATGUICREATEEDIT

The ATGUICREATEEDIT routine creates an *edit 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 use types.

Calling syntax:

```
CALL ATGUICREATEEDIT(APPID, FORMID, CTRLID,
        PARENTID, EVENTMASK, STYLE, LEFT, TOP,
        WIDTH, HEIGHT, ERRORS, 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:

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

Other arguments: for internal use - do not modify

GUISTATE

Properties supported by this item:

GPFORECOLOR	foreground (text) color (default is form's FORECOLOR)
GPBACKCOLOR	background color (default color is WindowBackground)
GPFONTNAME	name of default font (default is form's FONTNAME)
GPFONTSIZE	font size in points (default is form's FONTSIZE)
GPFONTBOLD	non-zero to use boldface font
GPFONTITALIC	non-zero to use italic font
GPENABLED	non-zero if control is enabled
GPVISIBLE	non-zero if control is visible
GPHELPID	help topic ID in the application help file
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
GPALIGN	0 = left, 1 = right, 2 = center
GPDEFVAL	default value of control (also sets current value)
GPVALUE	value of control

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

GECHANGE	the user changed the control's value – this event is fired for each change (eg 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 – use the ARGS<1,1> argument to determine the ID of the control triggering the event, ARGS<2> is the current value
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
GEDEACTIVATE	the control is no longer the active control

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.

Calling syntax:

```
CALL ATGUICREATELABEL(APPID, FORMID, CTRLID,
     PARENTID, EVENTMASK,  CAPTION, LEFT, TOP,
     WIDTH, HEIGHT, ERRORS, 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:

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

Other arguments: for internal use - do not modify

GUISTATE

Properties supported by this item:

GPFORECOLOR	foreground (text) color (default is form's FORECOLOR)
GPBACKCOLOR	background color (default is form's 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 (default); 1 = flat border; 2 = 3D border
GPALIGN	0 = left, 1 = right, 2 = center
GPDEFVAL	default value of control (also sets current value)
GPVALUE	value of control

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

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) item(s) 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 "drops down" so that items may be selected. Thus the height of the *list control* is only the height of a single item if the drop-down style is chosen; otherwise the height is the height of the displayed list.

Calling syntax:

```
CALL ATGUICREATELIST(APPID, FORMID, CTRLID,
        PARENTID, EVENTMASK, STYLE, LEFT, TOP,
        WIDTH, HEIGHT, ERRORS, 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:

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

Other arguments: for internal use - do not modify

GUISTATE

Properties supported by this item:

GPFORECOLOR	foreground (text) color (default is form's FORECOLOR)
GPBACKCOLOR	background color (default color is WindowBackground)
GPFONTNAME	name of default font (default is form's FONTNAME)
GPFONTSIZE	font size in points (default is form's FONTSIZE)
GPFONTBOLD	non-zero to use boldface font
GPFONTITALIC	non-zero to use italic font
GPENABLED	non-zero if control is enabled
GPVISIBLE	non-zero if control is visible
GPHELPID	help topic ID in the application help file
GPBORDER	0 = no border; 1 = flat border; 2 = 3D border (default)
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	list (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.
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.
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	value of control (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 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 – use the ARGS<1,1> argument to determine the ID of the control triggering the event, ARGS<2> is the current value
GECLICK	the user clicked the left mouse button on this control (a click event is also triggered by selecting an item using the cursor keys)
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
GEDEACTIVATE	the control is no longer the active control

ATGUICREATECOMBO

The ATGUICREATECOMBO routine creates a *combo control*. A *combo control* is a combination of *edit* and *list controls*. A *combo control* is used to select an item from a list of items, or enter text directly. The *value* of a *combo control* is the text string of the edit portion of the control. The *combo control* can display multiple columns, 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 a row in the drop-down list, the contents of the designated "data" column (of the selected row) are copied to the edit portion of the control.

Calling syntax:

```
CALL ATGUICREATECOMBO(APPID, FORMID, CTRLID,
     PARENTID, EVENTMASK, LEFT, TOP, WIDTH,
     HEIGHT,  ERRORS, 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:

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

Other arguments: for internal use - do not modify

GUISTATE

Properties supported by this item:

GPFORECOLOR	foreground (text) color (default is form's FORECOLOR)
GPBACKCOLOR	background color (default color is WindowBackground)
GPFONTNAME	name of default font (default is form's FONTNAME)
GPFONTSIZE	font size in points (default is form's FONTSIZE)
GPFONTBOLD	non-zero to use boldface font
GPFONTITALIC	non-zero to use italic font
GPENABLED	non-zero if control is enabled
GPVISIBLE	non-zero if control is visible
GPHELPID	help topic ID in the application help file
GPBORDER	0 = no border; 1 = flat border; 2 = 3D border (default)
GPDATATYPE	specify one of the enumerated data types
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	list (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.
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.
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 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 (eg 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 – use the ARGS<1,1> argument to determine the ID of the control triggering the event, ARGS<2> is the current value
GECLICK	the user clicked the left mouse button on this control (a click event is also triggered by selecting an item using the cursor keys)
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
GEDEACTIVATE	the control is no longer the active control

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.

Calling syntax:

CALL ATGUICREATEOPTION(APPID, FORMID, CTRLID, PARENTID, EVENTMASK, CAPTION, ITEMS, LEFT, TOP, WIDTH, HEIGHT, ERRORS, 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)
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:

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

Other arguments: for internal use - do not modify

GUISTATE

Properties supported by this item:

GPFORECOLOR	foreground (text) color (default is form's FORECOLOR)
GPBACKCOLOR	background color (default is form's 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
GPHELPID	help topic ID in the application help file
GPBORDER	0 = no border; 1 = flat border; 2 = 3D border (default)
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). An individual item 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

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 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 – use the

	ARGS<1,1> argument to determine the ID of the control triggering the event, ARGS<2> is the current value
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)
GECONTEXT	the user clicked the right mouse button on this control
GEACTIVATE	the control has become the active control
GEDEACTIVATE	the control is no longer the active control

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).

Calling syntax:

CALL ATGUICREATECHECK(APPID, FORMID, CTRLID, PARENTID, EVENTMASK, CAPTION, LEFT, TOP, WIDTH, HEIGHT, ERRORS, 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
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:

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

Other arguments: for internal use - do not modify

GUISTATE

Properties supported by this item:

GPFORECOLOR	foreground (text) color (default is form's FORECOLOR)
GPBACKCOLOR	background color (default is form's 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
GPHELPID	help topic ID in the application help file
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):

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 – use the ARGS<1,1> argument to determine the ID of the control triggering the event, ARGS<2> is the current value
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
GECONTEXT	the user clicked the right mouse button on this control
GEACTIVATE	the control has become the active control

GEDEACTIVATE	the control is no longer the active control

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*.

Calling syntax:

CALL ATGUICREATEBUTTON(APPID, FORMID, CTRLID,
PARENTID, EVENTMASK,
CAPTION, LEFT, TOP, WIDTH, HEIGHT, ERRORS,
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
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:

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

Other arguments: for internal use - do not modify

GUISTATE

Properties supported by this item:

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
GPHELPID	help topic ID in the application help file
GPPICTURE	optional picture filename or URL for graphical buttons (don't 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

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. 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).

Calling syntax:

```
CALL ATGUICREATEGRID(APPID, FORMID, CTRLID,
        PARENTID, EVENTMASK, STYLE, COLUMNS,
        COLHEADING, LEFT, TOP, WIDTH, HEIGHT,
        ERRORS, 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:

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

Other arguments: for internal use - do not modify

GUISTATE

Properties supported by this item:

GPFORECOLOR	foreground (text) color (default is form's FORECOLOR); individual column, row or cell colors may be updated by specifying non-zero values for the COL and ROW parameters
GPBACKCOLOR	background color (default color is WindowBackground) ; individual column, row or cell colors may be updated by specifying non-zero values for the COL and ROW parameters
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 in the application help file
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
GPCOLUMNS	number of columns in the grid
GPCOLHEADING	list (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 types with value marks)
GPCOLFIELDTYPE	0 = label, 1 = edit, 2 = check box, 3 = drop-down list, 4 = drop-down combo, 5 = label with ellipsis button
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 multi-column drop-down list, this is the "data" column of the drop-down list (separate each column's setting with value marks; non-drop-down columns 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; separate individual list items with subvalue marks; separate columns within the list with vertical bars (|), separate each grid column's list with value marks; non-drop-down columns are ignored. The list for an individual column can be accessed by ATGUIGETPROP and

	ATGUISETPROP by specifying non-zero COL argument.
GPCOLUMN	current grid column
GPROW	current grid row
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; use the ARGS argument to determine the column (ARGS<1,1>) and row (ARGS<1,2>) of the cell clicked.
GECONTEXT	the user clicked the right mouse button on this control; use the ARGS argument to determine the column (ARGS<1,1>) and row (ARGS<1,2>) of the cell clicked.
GEELLIPSIS	the user clicked the ellipsis button (…) in a cell; use the ARGS argument to determine the column (ARGS<1,1>) and row (ARGS<1,2>) triggering the event
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 – use the ARGS<1,1> argument to determine the ID of the control triggering the event, ARGS<2> is the current value (entire grid) – this event only valid if grid is editable
GEVALIDATECELL	the user finished editing contents of a cell; use the ARGS argument to determine the column (ARGS<1,1>)

	and row (ARGS<1,2>) just edited – ARGS<2> is the new cell value – this event only valid if grid is editable
GEVALIDATEROW	the user has changed the current row in the grid; use the ARGS<1,1> argument to determine the row which was previously active – ARGS<2> contains the previously active row's values separated by sub-value marks – this event only valid if grid is editable
GEACTIVATECELL	the user has moved to a new cell; use the ARGS argument to determine the new column (ARGS<1,1>) and row (ARGS<1,2>)
GEACTIVATE	the control has become the active control
GEDEACTIVATE	the control is no longer the active control

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, ERRORS, 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
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:

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

Other arguments: for internal use - do not modify

GUISTATE

Properties supported by this item:

GPENABLED	non-zero if control is enabled
GPVISIBLE	non-zero if control is visible
GPBORDER	0 = no border; 1 = flat border; 2 = 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

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

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.

Calling syntax:

CALL ATGUICREATEFRAME(APPID, FORMID, CTRLID, PARENTID, EVENTMASK, CAPTION, LEFT, TOP, WIDTH, HEIGHT, ERRORS, GUISTATE)

Input arguments:

APPID	identifies which application control belongs to
FORMID	identifies which form control belongs to
PARENTID	identifies a "parent" container control, such as a frame, if any
CTRLID	control identifier
EVENTMASK	list of events this control responds to (must include GECLICK)
CAPTION	caption text
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:

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

Other arguments: for internal use - do not modify

GUISTATE

Properties supported by this item:

GPFORECOLOR	foreground (text) color (default is form's FORECOLOR)
GPBACKCOLOR	background color (default is form's 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

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, ERRORS, 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:

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

Other arguments: for internal use - do not modify

GUISTATE

Properties supported by this item:

GPENABLED	non-zero if control is enabled
GPVISIBLE	non-zero if control is visible

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

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*.

Calling syntax:

CALL ATGUICREATETAB(APPID, FORMID, CTRLID, PARENTID, EVENTMASK, CAPTION, ERRORS, 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	tab caption text

Output arguments:

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

Other arguments: for internal use - do not modify

GUISTATE

Properties supported by this item:

GPFORECOLOR	foreground (text) color (default is form's FORECOLOR)

GPBACKCOLOR	background color (default is form's BACKCOLOR)
GPENABLED	non-zero if control is enabled
GPVISIBLE	non-zero if control is visible
GPCAPTION	caption text

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

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 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, ERRORS, 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 7 fields separated by subvalue marks

Output arguments:

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

Other arguments: for internal use - do not modify

GUISTATE

Menu item record format:

subvalue 1	menu item ID (unique within menu), use a "-" to indicate a separator between menu items: special item IDs may be used for certain standard operations: *CUT, *COPY, *PASTE, *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 numeric library picture ID or filename)
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)

Properties supported by this item:

GPVISIBLE	non-zero if menu is visible
GPITEMS	menu item records separated by value marks - each menu item record contains 7 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 – the event arguments ARGS<1,1> specifies 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*.

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, ERRORS, 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 6 fields separated by subvalue marks

Output arguments:

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

Other arguments: for internal use - do not modify

GUISTATE

Toolbar item record format:

subvalue 1	tool 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, *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	reserved – use null
subvalue 3	tooltip help text
subvalue 4	button picture (either numeric library picture ID or filename)
subvalue 5	non-zero if item is enabled
subvalue 6	non-zero if item is visible

Properties supported by this item:

GPVISIBLE	non-zero if menu is visible
GPITEMS	toolbar item records separated by value marks - each toolbar item record contains 6 fields separated by subvalue marks. An individual item can be accessed by ATGUIGETPROP and ATGUISETPROP by specifying non-zero ROW argument.
GPALIGN	0 = floating, 1 = top, 2 = bottom, 3 = left, 4 = right
GPSTYLE	0 = small icons, 1 = large icons

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 – the event arguments ARGS<1,1> specifies 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, ERRORS,
     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:

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

Other arguments: for internal use - do not modify

GUISTATE

ATGUIACTIVATE

The ATGUIACTIVATE function is called activate a *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,
      ERRORS, GUISTATE)
```

Input arguments:

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

Output arguments:

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

Other arguments: for internal use - do not modify

GUISTATE

ATGUISETPROP

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

Calling syntax:

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

Input arguments:

APPID	application identifier
FORMID	form identifier (null if setting application property)
CTRLID	control identifier (null if setting form or application property)
PROPERTY	property code (see each control or form topic for specific properties; property codes begin with GP without the dot, eg. GPFORECOLOR)
COL	if property accepts multiple columns, specifies column number; zero indicates all columns
ROW	if property accepts multiple rows, specifies row number; zero indicates all rows
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:

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

Other arguments: for internal use - do not modify

GUISTATE

ATGUIGETPROP

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

Calling syntax:

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

Input arguments:

APPID	application identifier
FORMID	form identifier (null if setting application property)
CTRLID	control identifier (null if setting form or application property)
PROPERTY	property code (see each control or form topic for specific properties; property codes begin with GP without the dot, eg. GPFORECOLOR)
COL	if property accepts multiple columns, specifies column number; zero indicates all columns
ROW	if property accepts multiple rows, specifies row number; zero indicates all rows

Output arguments:

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

Other arguments: for internal use - do not modify

GUISTATE

ATGUILOADVALUES

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

Calling syntax:

```
CALL ATGUILOADVALUES(APPID, FORMID, CTRLIDS,
      VALUES, ERRORS, 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:

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

Other arguments: for internal use - do not modify

GUISTATE

ATGUIGETVALUES

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

Calling syntax:

```
CALL ATGUIGETVALUES(APPID, FORMID, CTRLIDS,
      VALUES, ERRORS, 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
ERRORS	ERRORS<1> is non-zero if errors have occurred (see ERRORS topic for details)

Other arguments: for internal use - do not modify

GUISTATE

ATGUIGETUPDATES

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

Calling syntax:

```
CALL ATGUIGETUPDATES(APPID, FORMID, CTRLIDS,
      VALUES, ERRORS, 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
ERRORS	ERRORS<1> is non-zero if errors have occurred (see ERRORS topic for details)

Other arguments: for internal use - do not modify

GUISTATE

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.

Calling syntax:

CALL ATGUIRESET(APPID, FORMID, ERRORS, GUISTATE)

Input arguments:

APPID	application identifier
FORMID	form identifier

Output arguments:

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

Other arguments: for internal use - do not modify

GUISTATE

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, ERRORS,
     GUISTATE)
```

Input arguments:

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

Output arguments:

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

Other arguments: for internal use - do not modify

GUISTATE

ATGUIINSERT

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

Calling syntax:

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

Input arguments:

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

Output arguments:

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

Other arguments: for internal use - do not modify

GUISTATE

ATGUIREMOVE

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

Calling syntax:

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

Input arguments:

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

Output arguments:

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

Other arguments: for internal use - do not modify

GUISTATE

ATGUISHOW

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

Calling syntax:

```
CALL ATGUISHOW(APPID, FORMID, CTLID, MENUID,
      ERRORS, 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:

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

Other arguments: for internal use - do not modify

GUISTATE

ATGUIHIDE

The ATGUIHIDE routine makes a form or control invisible.

Calling syntax:

```
CALL ATGUIHIDE(APPID, FORMID, CTLID, MENUID,
      ERRORS, 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:

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

Other arguments: for internal use - do not modify

GUISTATE

ATGUIENABLE

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

Calling syntax:

```
CALL ATGUIENABLE(APPID, FORMID, CTLID, MENUID,
      ERRORS, 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:

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

Other arguments: for internal use - do not modify

GUISTATE

ATGUIDISABLE

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

Calling syntax:

```
CALL ATGUIDISABLE(APPID, FORMID, CTLID, MENUID,
        ERRORS, 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:

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

Other arguments: for internal use - do not modify

GUISTATE

ATGUIPRINT

The ATGUIPRINT routine prints a form.

Calling syntax:

CALL ATGUIPRINT(APPID, FORMID, ERRORS, GUISTATE)

Input arguments:

APPID	application identifier
FORMID	form identifier

Output arguments:

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

Other arguments: for internal use - do not modify

GUISTATE

ATGUIWAITEVENT

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

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

Calling syntax:

```
CALL ATGUIWAITEVENT(APPID, FORMID, CTLID,
        EVENT, ARGS, ERRORS, GUISTATE)
```

Output arguments:

APPID	application identifier
FORMID	form identifier
CTRLID	control (or menu) identifier which triggered the event
EVENT	event code (see EVENT documentation for each control type)
ARGS	any event argument values separated by value marks (see EVENT documentation for each control type)

Output arguments (both):

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

Other arguments: for internal use - do not modify

GUISTATE

ATGUICHECKEVENT

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

Calling syntax:

```
CALL ATGUICHECKEVENT(TIMEOUT, APPID, FORMID,
        CTLID, EVENT, ARGS, ERRORS, GUISTATE)
```

Output arguments:

TIMEOUT	timeout value in milliseconds
APPID	application identifier
FORMID	form identifier
CTRLID	control (or menu) identifier which triggered the event
EVENT	event code (see EVENT documentation for each control type)
ARGS	any event argument values separated by value marks (see EVENT documentation for each control type)

Output arguments (both):

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

Other arguments: for internal use - do not modify

GUISTATE

ATGUIPOSTEVENT

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

Calling syntax:

```
CALL ATGUIPOSTEVENT(APPID, FORMID, CTLID, EVENT,
        ARGS, POSITIONERRORS, GUISTATE)
```

Output arguments:

APPID	application identifier
FORMID	form identifier
CTRLID	control (or menu) identifier which triggered the event
EVENT	event code (see EVENT documentation for each control type)
ARGS	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 (both):

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

Other arguments: for internal use - do not modify

GUISTATE

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 routines 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, ERRORS, GUISTATE)
CALL ATGUIENDMACRO(MACRO, ERRORS, 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):

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

Other arguments: for internal use - do not modify

GUISTATE

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* 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. Using *macros* and *templates* a GUI-ized application can simply read the *macro* or *template* item from a file and call ATGUIRUNMACRO.

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, ERRORS, GUISTATE)

Input arguments:

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

Output arguments:

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

Other arguments: for internal use - do not modify

GUISTATE

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, ERRORS, GUISTATE)
```

Input arguments:

PRMPT	custom prompt message (multiple lines as OK; separate lines with CR/LF)
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
ERRORS	ERRORS<1> is non-zero if errors have occurred (see ERRORS topic for details)

Other arguments: for internal use - do not modify

GUISTATE

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, ERRORS,
      GUISTATE)
```

Input arguments:

PRMPT	custom prompt message (multiple lines as OK; separate lines with CR/LF)
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
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
ERRORS	ERRORS<1> is non-zero if errors have occurred (see ERRORS topic for details)

Other arguments: for internal use - do not modify

GUISTATE

ATGUIOPENDIALOG
ATGUISAVEDIALOG

The ATGUIOPENDIALOG and ATGUISAVEDIALOG routines are called to prompt for a file name to open or save. A standard “file open” or “file save as” dialog box is displayed with a custom caption. If the user clicks the Cancel button, the returned value is null.

Calling syntax:

CALL ATGUIOPENDIALOG(CAPTION, DEFAULT, FILTER, VALUE, ERRORS, GUISTATE)

CALL ATGUISAVEDIALOG(CAPTION, DEFAULT, FILTER, VALUE, ERRORS, 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:*.*”

Output arguments:

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

Other arguments: for internal use - do not modify

GUISTATE

ATGUIGETACTIVE

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

Calling syntax:

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

Input arguments:

none

Output arguments:

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

Other arguments: for internal use - do not modify

GUISTATE

ATGUISUSPEND
ATGUIRESUME

The ATGUISUSPEND routine suspends the currently running GUI applications and allows normal terminal function. ATGUIRESUME resumes the suspended GUI applications. The GUISTATE variable must be maintained between the ATGUISUSPEND and ATGUIRESUME calls. The GUI application is not hidden while suspended, but it is frozen.

Calling syntax:

CALL ATGUISUSPEND(ERRORS, GUISTATE)

CALL ATGUIRESUME(ERRORS, GUISTATE)

Input arguments:

GUISTATE	ATGUIRESUME only

Output arguments:

GUISTATE	ATGUISUSPEND only
ERRORS	ERRORS<1> is non-zero if errors have occurred (see 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:

GPENTMASK	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 – forms are relative to the screen, controls are relative to their container control or form
GPWIDTH	width of the form or control
GPHEIGHT	height of the form or control
GPCHANGED	zero if control's value has not been updated by the user, otherwise one – the changed property of a container, such as a form, reflects the changed status of any contained control – that is if any control's changed status is set, the containing form's changed status is also set – you can test the changed status of a form to determine if any controls within the form have been updated

ERRORS

ERRORS is a structure (dynamic array) returned as a result of all ATGUI… routines. The ERRORS 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 ERRORS structure, ERRORS<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 ERRORS structure. Each error (attribute) consists of 6 fields (separated by value marks).

ERRORS<x,1> = error severity

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

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

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

ERRORS<x,5> = error number

ERRORS<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.

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 Pick/BASIC code is implemented as a subroutine, rather than as a standalone program.

In order to avoid stressing the Windows client system, you can manage which sub-applications are loaded at any given moment. Several pseudo-events are used by the sub-application subroutines to load and unload the sub-application, and show and hide forms within the sub-application. The pseudo-events are the string literals ‘LOAD’, ‘UNLOAD’, ‘SHOW’, and ‘HIDE’. To start the first sub-application, the master application can use the ATGUIPOSTEVENT to post a load event for the first sub-application. Then, the standard event loop will dispatch the event to the correct sub-application. The default action for the load event is to load the sub-application. It is also useful to show the first form for the sub-application in the load event handler.

By default when a form receives a Close event, the form is hidden. 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.

OBJECT BRIDGE

Object Bridge is a Windows Object extension (ActiveX Automation) for Pick/BASIC programs. This allows a Pick/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 Pick/BASIC interface implemented as a set of callable subroutines, a “remote procedure call” (RPC) conduit (AccuTerm 2000 session) and a Windows object manager (installed with AccuTerm 2000).

Object Bridge allows a Pick/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 Pick/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 2000 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.

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 dereived 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.

Getting Object Properties

```
ATGETPROPERTY(OBJID, PROPNAMES, PROPVALUES,
ERRMSG, OPTS)
```

Use this routine to set 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 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. Releasign 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 specified 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` 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 `[Custom]` section of the ATWIN.INI file. The entry is:

```
...
[Custom]
MouseTable=filename
...
```

The Mouse Pattern Table file (***filename***) 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 DOS editor does not preserve tab characters when saving a file, so edit this file using Notepad instead!

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
R	0	U	8
P	1	D	9
j	2	T	:
O	3	@	;
S	4	]	<
i	5	Q	=
C	6	J	>
B	7	A	?

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
j	@ or A or B or C
O	D or E or F or G
P	H or I or J or K
i	L or M or N or O
R	P or Q or R or S
D	T or U or V or W
S	X or Y or Z or [
Q	\ or] or ^ or _
T	` or a or b or c
C	d or e or f or g
U	h or i or j or k

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***

Viewpoint Key Code Table (continued)

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 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		
SHIFT+INS		
DEL	**DEL**	***DEL***
SHIFT+DEL	**DEL LINE**	***DEL***
HOME	**HOME**	***CSI*** H
SHIFT+HOME	**SHIFT+HOME**	***CSI*** H
END		
SHIFT+END		
PGUP	**PGUP**	***CSI*** V
SHIFT+PGUP	**SHIFT+PGUP**	***CSI*** V
PGDN	**PGDN**	***CSI*** U
SHIFT+PGDN	**SHIFT+PGDN**	***CSI*** U
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.

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*** 1 7 ~
F7	**F7**	***CSI*** 1 8 ~
F8	**F8**	***CSI*** 1 9 ~
F9	**F9**	***CSI*** 2 0 ~
F10	**F10**	***CSI*** 2 1 ~
F11	**F11**	***CSI*** 2 3 ~
F12	**F12**	***CSI*** 2 4 ~
CTRL+F1	**F11**	***CSI*** 2 3 ~
CTRL+F2	**F12**	***CSI*** 2 4 ~
CTRL+F3	**F13**	***CSI*** 2 5 ~
CTRL+F4	**F14**	***CSI*** 2 6 ~
CTRL+F5	**F15**	***CSI*** 2 8 ~
CTRL+F6	**F16**	***CSI*** 2 9 ~
CTRL+F7	**F17**	***CSI*** 3 1 ~
CTRL+F8	**F18**	***CSI*** 3 2 ~
CTRL+F9	**F19**	***CSI*** 3 3 ~
CTRL+F10	**F20**	***CSI*** 3 4 ~

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 2000 installation program will install vendor supplied files when placed in the VARFILES directory of the last diskette. Files found in this directory will be copied into the target ATWIN 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. Files found in VARFILES\HOME will be copied to the "home" AccuTerm/Windows directory (usually this is the same as the target ATWIN directory, but in a network installation, the ATWIN directory may be shared, but the "home" directory will be private). This is where a customized ATWIN.INI, .ATLY or .ATCF file should be located.

The AccuTerm 2000 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 2000 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 setup program. The SETUP.INI file can be used to specify default values for several installation prompts:

```
[Defaults]
Maindir= program installation directory
Homedir= users home directory
Group= start menu group
Options=A
```

The `Maindir` is the directory where the program files are installed. If this is on a network share, specify the `Homedir` as a directory where the user’s configuration files will be stored. The `Group` is the start menu

group for the shortcut to start AccuTerm. The `Options=A` is included if you want the enhanced image support loaded (omit this line if standard image support is desired).

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 and Title

It is possible to use a custom icon and / or window title with AccuTerm 2000. To change the icon, add an `Icon` item to the `[Custom]` section of ATWIN.INI. To change the window title, add a `Title` item:

```
...
[Custom]
Icon=filename
Title=new window title
...
```

Custom Hot Key Blocking

It is possible to block the action of any hot keys, except those hot keys which open a top-level menu (use the menu designer to customize the menu if you need to modify this action). To enable this feature, add a `BlockedHotKeys` item to the `[Custom]` section of ATWIN.INI. Assign the list of hot keys to block to this item. For example, to block the **ALT**+P (printer toggle) hot keys, add `BlockedHotKeys=P` to the `[Custom]` section of ATWIN.INI.

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*** `; 0u` 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