



HostAccess Developer's Guide

Disclaimer

Every effort has been made to ensure that the information contained within this publication is accurate and up-to-date. However, Rogue Wave Software, Inc. does not accept liability for any errors or omissions.

Rogue Wave Software, Inc. continuously develops its products and services. We therefore reserve the right to alter the information within this publication without notice. Any changes will be included in subsequent editions of this publication.

As the computing industry lacks consistent standards, Rogue Wave Software, Inc. cannot guarantee that its products will be compatible with any combination of systems you choose to use them with. While we may be able to help, you must determine for yourself the compatibility in any particular instance of Rogue Wave Software, Inc. products and your hardware/software environment.

Rogue Wave Software, Inc. acknowledges that certain proprietary programs, products or services may be mentioned within this publication. These programs, products or services are distributed under Trademarks or Registered Trademarks of their vendors and/or distributors in the relevant country.

Your right to copy this publication, in either hard-copy (paper) or soft-copy (electronic) format, is limited by copyright law. You must obtain prior authorization from Rogue Wave Software, Inc. before copying, adapting or making compilations of this publication.

HostAccess is a trademark of Quovadx Ltd in the United Kingdom and is a registered trademark in the USA. Microsoft is a registered trademark and Windows is a trademark of the Microsoft Corporation. Other brands and their products are trademarks or registered trademarks of their respected holders and should be noted as such.

Copyright © 1993-2013 Rogue Wave Software, Inc.

Contents

DISCLAIMER	2
CONTENTS	3
INTRODUCTION	5
What is HostAccess?	6
How to use HostAccess	7
AIF TOOLKIT	9
Escape Sequence Summary	10
Using escape sequences	12
Conventions used	13
Sculpting the Screen	15
Managing Controls	20
Copy and Paste	27
Root Control Features	30
Secondary Windows	32
Buttons	37
String Lists	43
Combo Boxes	49
List Boxes	55
Manipulating a List Box	60
Edit Boxes	62
Validated Edit Boxes	69
Static Labels	74
Status Bar	78
Commands for menus, toolboxes and toolbars	80
Toolbars and Toolboxes	84
Menus	87
Changing Fonts	89
Invoking Windows Help	90
Timed Events	91
ActiveX (COM) Integration	92
Common Problems	101
AIF UTILITIES	103
How AiF Sequences Work	103
Sequences Summary	105
Tailoring the Environment	110
Using Windows	114
Using AiF menus	117
Using Selection Boxes	132
Developer's Guide	3

Using Field Input	137
Box Input	142
Save Environment	147
Display Optimisation	148
FORMs	151
Freeze On/Off	154
Host Echo On/Off	156
Applications Enhancement	157
Using Macros	169
Keyboard Control Features	170
Mouse Control	179
Programmable DOS Gateway	182
DOS Keyboard Stacker	184
Printing to a DOS File or Device	187
Data Extraction to DOS and Windows	193
Displaying Images	194
Miscellaneous AiF Facilities	200
DYNAMIC DATA EXCHANGE	206
How DDE Works	206
DDE Sequences: Summary	207
Using DDE with HostAccess	207
DDE Client Support	208
DDE Server Support	210
USING THE MACRO LANGUAGE	213
Syntax Conventions	213
Using AiF Escape Sequences	214
Declaring Variables	214
Using Functions	215
Using Procedures	218
DESCRIBING IMAGES	235
Image Types	235
Defining a simple image	237
Defining Labelled Images	241
Inbuilt Labelled Images	243
Defining Button Windows for Images	244

Introduction

Welcome to the HostAccess Developer's Guide. This book is designed to help you to fully exploit HostAccess in applications development and is divided into the following chapters:

Chapter	Title	Topics covered
Chapter 1	Introduction.	What is HostAccess and how to make the best use of it.
Chapter 2	AiF TOOLKIT.	How to use HostAccess's Application interface Facility (AiF) to develop Graphical User Interface (GUI)-like applications.
Chapter 3	AiF Utilities.	How to use the AiF for screen manipulation and DOS library routines, for example, opening and closing windows, loading menus, controlling printing.
Chapter 4	Dynamic Data Exchange.	How to use HostAccess as a DDE client and as a DDE server.
Chapter 5	The Macro Language.	How to use HostAccess's powerful macro language.
Appendix A	Describing Images.	How you can describe button images in detail using Windows AiF escape sequences.

What is HostAccess?

HostAccess is really three solutions in one.

Using HostAccess as a terminal emulator, you can achieve connectivity immediately and then enhance your screen using the autoGUI and autosculpture functions. These features are described in HostAccess's User Guide.

As a desktop integrator, HostAccess brings data from host applications into your familiar Windows- and DOS-based spreadsheets, word processors and other programs.

As a rejuvenation tool you can use HostAccess to transform your host applications quickly and easily with its extensive toolkit known as the Applications interface Facility (AiF). This has been designed to enable host developers to rejuvenate their legacy applications without needing to totally re-engineer them.

Desktop integration and AiF facilities are described in this Developer's Guide. If you are a PICK user, you can also make use of an extensive set of subroutines documented in the PICK Guide, please contact your dealer for further information.

Typical terminal screen before AutoGUI

03/31/96		Telesales System		Fruit and Vegetable Account	
Product	APPLES	Stock	10%	New Orders	10M
Last Order	1st Dec 96	Shipping	Air		
Outstanding Order	2400				
Supplier	FAST FRUITS	Description	Bright Red Apples shiny and soft to touch. Sells well on supermarket shelves when displayed predominantly.		
Country of Origin	AFRICA		xref APP017 for alternatives.		
Cost Per KG	1231.12				
Carrier	FAST FRUITS				
F1 - Product F2 - Suppliers F3 - Carriers F8 - Save F9 - Exit F10 - Help :					

Terminal screen with AutoGUI

03/31/96		Telesales System		Fruit and Vegetable Account	
Product	APPLES	Stock	10%	New Orders	10M
Last Order	1st Dec 96	Shipping	Air		
Outstanding Order	2400				
Supplier	FAST FRUITS	Description	Bright Red Apples shiny and soft to touch. Sells well on supermarket shelves when displayed predominantly.		
Country of Origin	AFRICA		xref APP017 for alternatives.		
Cost Per KG	1231.12				
Carrier	FAST FRUITS				
F1 - Product F2 - Suppliers F3 - Carriers F8 - Save F9 - Exit F10 - Help :					

How to use HostAccess

HostAccess's 3-stage process allows you to implement your unique IT strategy at your own pace. Begin by creating a tailored GUI, shaping a Windows look and feel for host applications.

The next phase is to integrate your host applications into your standard desktop applications. Data can be downloaded into any standard spreadsheet and text into any word processor. At this stage, you are enhancing your host applications using Windows' Dynamic Data Exchange (DDE) and OLE automation. HostAccess's tight integration of host data with your application server makes data available to each desktop.

Finally, transform your host applications, changing the way they look, feel and respond. HostAccess's tools give your host applications access to, and control of any desktop resources. Open up your host applications to any PC clients, including COM automation objects and ActiveX controls.

Creating a Windows look and feel

HostAccess's AiF toolkit allows you to give applications the same GUI as your familiar desktop. Give your host applications any Windows control, including secondary windows, icons and buttons. Host applications can be made faster, clearer and even fully event driven.

AiF is a library of ANSI-compliant escape sequences sent from the host to HostAccess using normal terminal display functions in the host application.

AiF's phased approach puts you in control of the speed and sophistication of your development programme. In addition, you can drive this development with your current host application design skills, using existing programming languages from COBOL and Basic to FORTRAN and SQL.

Desktop integration

Integrate your host applications into your standard desktop applications using DDE, a standard way of communicating between Windows applications. You can use DDE to use HostAccess as a DDE client to Windows applications giving your host applications almost total control over any other Windows product. You can also use HostAccess as a DDE server - this means you can write Windows programs in applications such as Word or Excel which can send or receive data to and from the host. This is described in Chapter 4, Dynamic Data Exchange.

Additionally, HostAccess's file transfer facilities puts your data where it's most functional to users, creating a genuine two-way environment.

AiF TOOLKIT

This chapter describes how you can make full use of HostAccess to create a Windows look and feel for your host applications. First, you need to use HostAccess as a terminal emulator to run your host applications on your PC. Initially your host applications may continue to look and work as they have for years as they do on your old terminals

You can then choose the available GUI functions to transform your terminal screen. HostAccess allows you to transform the terminal screen itself, allowing you to completely transform the application's look and feel.

A wide range of functions known as the **Applications interface Facility (AiF)** are available to all host developers which enables you to use these features to create a true Windows GUI appearance for your host applications, with only a minimal amount of coding.

You can use HostAccess's Windows AiF to create and use:

- Push buttons.
- Radio buttons.
- Check boxes
- Edit boxes.
- List boxes.
- Combo boxes.
- Secondary windows.
- String lists.
- Commands.
- Menus.

You can use these in a fully interactive fashion, reacting to user input. For example, you can detect whenever the user clicks on a pushbutton, and react accordingly.

Escape Sequence Summary

Sculpture, see from page 15.

- ESC_1 Turns sculpture mode on or off.
- ESC_2 Draws a sculpted box.
- ESC_3 Draws a sculpted horizontal line.
- ESC_4 Draws a sculpted vertical line.
- ESC_5 Changing default colours.

Controls, see from page 20.

- ESC_9 Verifies a named control is valid.
- ESC_10 Destroys a named control.
- ESC_11 Enables/disables a named control.
- ESC_12 Shows/hides named control(s).
- ESC_13 Re-sizes and/or moves a control's window.
- ESC_14 Changes a control's colours.
- ESC_15 Sets/clears event reporting for named controls.
- ESC_16 Sets input focus to a named control.
- ESC_17 Sets input focus to an unknown control.
- ESC_18 Uses groups of controls.
- ESC_19 Returns a given string when an event occurs.
- ESC_20 Sets control's accelerator character.
- ESC_21 Sets the meaning of the <Return> key.

Copy and paste, see from page 27.

- ESC_22 Copies an area of the screen.
- ESC_23 Pastes a saved area of screen.
- ESC_24 Clears all slots of saved screen regions.

Root control, see from page 30.

- ESC_25 Creates the root control.
- ESC_26 Reads a value from the root control.
- ESC_27 Manipulate the root control.
- ESC_28 Miscellaneous control functions.

Secondary Windows, see from page 32.

- ESC_29 Secondary Windows manipulation.

Pushbuttons, check boxes and buttons, see from page 37.

- ESC_30 Creates a text pushbutton with a text label.
- ESC_31 Creates an image pushbutton.
- ESC_32 Display an image.
- ESC_33 Commands for toolboxes, toolbars, commands and menus.
- ESC_34 Creates a check box.
- ESC_35 Creates a radio button.
- ESC_36 Reads a value from a button.
- ESC_37 Manipulates a button.

Lists, comboboxes and edit boxes, see from page 55

- ESC_40 Creates/adds entries to/removes entries from string lists.
- ESC_41 Read a value from a string list.
- ESC_42 Manipulates a string list.
- ESC_45 Creates a list box or combo box.
- ESC_46 Reads a value from a list box or combo box.
- ESC_47 Manipulates a list box or combo box.
- ESC_50 Creates an edit box.
- ESC_51 Reads a value from an edit box.
- ESC_52 Manipulates an edit box.
- ESC_53 Creates a static label.
- ESC_70 Attach a validation to an edit box.

Miscellaneous

- ESC_56 Changes the Windows pointer
- ESC_91 Creates a modal message box.
- ESC_92 Sets status bar text.
- ESC_93 Changes text font.
- ESC_94 Status bar and Windows help functions.
- ESC_95 Returns notification after a set time.

Using escape sequences

To use AiF, you send **AiF escape sequences** from the host to the PC. An escape sequence enables you to send encoded signals to the host.

Any host process that can send output to a terminal can also use AiF by sending special escape sequences to HostAccess running on a PC. HostAccess intercepts these escape sequences and takes the appropriate action (for example, saving a screen image).

Software developers normally define these AiF escape sequences so that they can be referenced globally as variables by their applications code (either at run-time or compile time).

AiF escape sequences are standard ANSI X3.64 compliant escape sequences, belonging to the ANSI APC (Application Program Command) class of sequences.

To use AiF properly, you should be familiar with the concept of **controls**. Controls are Windows objects that are held in HostAccess's memory which have associated names (**control ids**).

Many Windows AiF escape sequences have control-id string parameters. Unless otherwise stated, you can assume that these parameters refer to the relevant control id as described here. A control-id is a unique identifier. You must define a control sequence before it can be used in an escape sequence.

Controls can be defined in a list and then sent to HostAccess from the host as a group.

Format of Escape Sequences

HostAccess expects AiF escape sequences to conform to a certain format. Every AiF escape sequence starts with the ESCape character (ASCII decimal value 27). Sequences take the following format:

```
ESC_nn ; Int1 ; Int2 ; ... Intn w String1 ; String2 ; ... Stringn ESC\
```

Where:

ESC	Is the escape character.
_	Is an underscore character. This can be modified if required.
nn	Is the number of the particular Windows AiF escape sequence you want to use.
Int1 ... Intn	Are integer parameters in the AiF escape sequence. These parameters depend on the AiF escape sequence and are always preceded by a delimiter. If the sequence has no integer parameters, there are no delimiters before the w character.

<code>;</code>	Is the default delimiter character, although it can be changed.
<code>w</code>	Is a literal 'w' character (signifying Windows). This must be lower-case.
String1 ... Stringn	Are string parameters in the AiF escape sequence. These strings depend on the AiF escape sequence and are separated by delimiters. Often, the first string will be the id of a control or object.
ESC\	Is the escape character, followed by <code>\</code> (a backslash character).

Delimiters are optional if their parameters are omitted. However, they are mandatory if used to indicate the order of a parameter.

For example, an AiF escape sequence has 3 optional parameters **x**, **y**, and **z**. You want to omit **x** and **y** from your sequence, using the default values. However, you also want to use the **z** parameter. Therefore, you must have 3 delimiters preceding the **z** parameter, to indicate its position.

Note: one of the most common programming errors when using AiF escape sequences is to forget or misplace the required delimiters

Conventions used

The following conventions are used when coding escape sequences:

- String and integer parameters may be optional depending on the escape sequence. Optional parameters are shown enclosed in braces - for example, `{; enable}`.
- Optional delimiters are also enclosed in braces.
- Default values for optional parameters are shown with asterisks. For example: `2* = do not enable` means that the relevant parameter takes the value 2 as a default.
- Control ids are case insensitive. For example, `OK.BUTTON` is the same as `ok.button`.
- Do not use spaces when coding the escape sequences. Spaces are shown in the escape sequence descriptions for clarity only.
- All AiF escape sequence parameters are given labels for clarity.
- The following applies when returning values to the host:
 - STX** Decimal value 02.
 - CR** Decimal value 13 (Carriage Return).

AiF Example

The following is a Windows AiF escape sequence:

```
ESC_1 {; enable} {; clear} w ESC\
```

This escape sequence has escape sequence number 1, takes two optional integer parameters, **enable** and **clear** and has no string parameters. It turns sculpture mode on/off, see page 16 for further information.

Screen Layout

For clarity, the positioning and drawing of the screen is performed in a grid method. The top left hand corner of any window has the grid co-ordinates of 1,1, and the bottom right can be 24, 80. Each character displayed upon the window takes up one cell, or grid row and column position.

Co-ordinates are given as **y, x**, where **y** is the number of vertical characters down the screen, and **x** is the number of characters across the screen.

Note: the top-left corner is position 1,1 - not position 0,0.

Many of the AiF escape sequences described in the following sections have **y** and **x** co-ordinates as parameters. Unless otherwise stated, you can assume that these parameters use the standard **y, x** system as described here.

Sculpting the Screen

You can use the following group of escape sequences to exploit the **sculpture** facilities of Windows:

- Turning sculpture mode on/off.
- Drawing sculpted boxes.
- Drawing sculpted lines.
- Changing default colours.

This allows you to create raised or sunken images on your screen, with the 3-D effect of a stone sculpture. A sculpted image is produced by shading sides of a picture, so when drawing a sculpted box, the top and left sides of the box are shaded one colour, and the bottom and right sides of the box are shaded another colour.

For example, to produce an image of a sunken box, you would need to shade the top/left sides a dark colour, and the bottom/right sides a light colour.

Because of the way HostAccess sculpting works, you can have a full sculpted screen without losing any of your 24 by 80 display. Sculpting works independently of your normal screen, so clearing the screen does not clear sculpture.

See page 19 for an example of using a macro to sculpt a screen.

Colours

Colours for sculpted boxes and lines are chosen from HostAccess's **colour palette**. This palette consists of colours 1 - 16 as follows:

Number	Colour	Number	Colour
1	black	9	dark grey
2	blue	10	light blue
3	green	11	light green
4	cyan	12	light cyan
5	red	13	light red
6	magenta	14	light magenta
7	brown	15	light brown
8	grey	16	white

When choosing colour, you can also choose the **clear** colour (number 17). This represents the colour of the current background, and has the effect of clearing the relevant lines and/or boxes.

Turning Sculpture Mode on/off

To turn sculpture mode on/off, use the following AiF escape sequence:

ESC_1 {; enable} {; clear} w ESC

Where:

enable 1 = disable sculpture mode.
2* = enable sculpture mode.

clear 0* = do not clear existing lines/boxes.
1 = clear existing sculpted lines and boxes.

Turning sculpture mode on or off does not affect the drawing of any sculpted boxes or lines. To draw a complete sculpted screen very quickly, draw your screen, then set sculpture mode to **on**.

You can also use this escape sequence to just clear sculpted lines and boxes, without switching mode. Clearing lines and boxes sets their border colour to **clear**.

Drawing Sculpted Boxes

To draw a sculpted box, use the following AiF escape sequence:

ESC_2 ; y ; x ; h ; wid {; col1} {; col2 ; col3} w ESC

Where:

y y co-ordinate of top of box.

x x co-ordinate of left of box.

h Height of box, in characters (rows).

wid Width of box, in characters.

col1 Colour selection:
1* = use default sculpture colours.
2 = use default colours, reversed (for raised instead of sunken appearance).
3 = use col2 and col3 parameters (below) to define colours.
4 = set to clear - clear the box from the screen.

col2 Palette colour of top and left sides of box. 1-17, ignored unless col1 = 3. See page 15 for a description of the colours.

col3 Colour of bottom and right sides of box. 1-17, ignored unless col1 = 3.

Example

To draw a box at (10, 2), height 5, width 10, and colours 1 (top/left) and 16 (bottom/right), use:

```
ESC_2 ; 10 ; 2 ; 5 ; 10 ; 3 ; 1 ; 16 w ESC \
```

Drawing Sculpted Lines

To draw a sculpted horizontal line, use the following AiF escape sequence:

```
ESC_3 ; y ; x ; len { ; col } w ESC \
```

Where:

- y** y co-ordinate of line origin.
- x** x co-ordinate of line origin.
- len** Length of line, in characters.
- col** Colour for line:
0* - default top/left colour.
1..17 - colour number

See Drawing sculptured boxes on page 16 to find out how to change the default and Colours on page 15 for a description of the colours.

To draw a sculpted vertical line, use the following AiF escape sequence:

```
ESC_4 ; y ; x ; len { ; col } w ESC \
```

Where **y**, **x**, **len** and **col** are as described above.

Examples

To draw a sculpted horizontal line at (12, 14), 10 characters (columns) long, with colour 1, use the following AiF escape sequence:

```
ESC_3 ; 12 ; 14 ; 10 ; 1 w ESC \
```

To draw a sculpted vertical line at (10, 31), 5 characters (rows) long, with colour 16, use the following AiF escape sequence:

```
ESC_4 ; 10 ; 31 ; 5 ; 16 w ESC \
```

Changing Default Colours

To change default sculpture colours, use the following AiF escape sequence:

```
ESC_5 ; top-left ; bot-right w ESC\
```

Where:

top-left Default top side and left side colour. 1..17: colour number. See Colours on page 15 for a description of the colours.

bot-right Default bottom side and right side colour. 1..17, as described above.

These colours will be used as defaults for all subsequent sculpted box and line drawing.

Example: Sculpted Drawing

The following diagram shows how you can use the sculpture features of the Windows AiF to produce lines and boxes. The example turns sculpture mode on, and draws three sculpted boxes, then a sculpted horizontal line (in a box), and a sculpted vertical line (in a box).

```
ESC_1 w ESC\
```

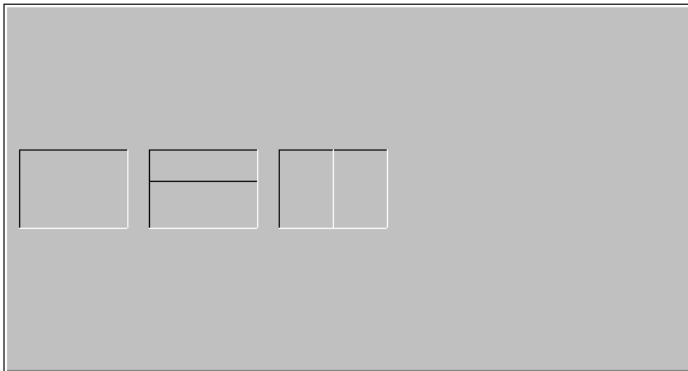
```
ESC_2 ; 10 ; 2 ; 5 ; 10 ; 3 ; 1 ; 16 w ESC\
```

```
ESC_2 ; 10 ; 14 ; 5 ; 10 ; 3 ; 1 ; 16 w ESC\
```

```
ESC_2 ; 10 ; 26 ; 5 ; 10 ; 3 ; 1 ; 16 w ESC\
```

```
ESC_3 ; 12 ; 14 ; 10 ; 1 w ESC\
```

```
ESC_4 ; 10 ; 31 ; 5 ; 16 w ESC\
```



Example of sculpting the screen using a macro

```

REM
REM Macro to demonstrate sculpted line drawing
REM using boxes that touch and those that do not.
REM Also using vertical and Horizontal lines to split boxes.
REM
REM For best results, select the NORMAL Attribute Colour
REM of Black foreground on Lightgrey background.
REM
REM Turn Sculpture mode ON.  print chr$(27) ; "_1w" ; chr$(27) ; "\";
REM Draw 3 Boxes .
print chr$(27) ; "_2;10;2;5;10;3;1;16w" ; chr$(27) ; "\"; print
chr$(27) ; "_2;10;14;5;10;3;1;16w" ; chr$(27) ; "\"; print chr$(27) ;
"_2;10;26;5;10;3;1;16w" ; chr$(27) ; "\";
REM Draw 2 Lines , Horizontal & Vertical
print chr$(27) ; "_3;12;14;10;1w" ; chr$(27) ; "\"; print chr$(27) ;
"_4;10;31;5;16w" ; chr$(27) ; "\";

REM Draw 4 Boxes directly underneath each other.
print chr$(27) ; "_2;10;40;1;30;3;1;16w" ; chr$(27) ; "\"; print
chr$(27) ; "_2;11;40;1;30;3;1;16w" ; chr$(27) ; "\"; print chr$(27) ;
"_2;12;40;1;30;3;1;16w" ; chr$(27) ; "\"; print chr$(27) ;
"_2;13;40;1;30;3;1;16w" ; chr$(27) ; "\"; print chr$(27) ;
"_2;14;40;1;30;3;1;16w" ; chr$(27) ; "\";
REM NOTES : Please note that each line has the ; at the end.
REM This will suppress the CRLF and stop the screen from REM
scrolling.

```

Managing Controls

These sections deal with general manipulation of specific controls, which have names given by **control-id** parameters.

You can create **control groups**, containing several controls. These control groups are created with specific names - **control group ids**.

Verify a Control

To verify that a named control has been created (i.e. verify the control-id is in use), use the following AiF escape sequence:

ESC_9 w control-id ESC

Where:

control-id Control id or group id.

This returns:

<STX> status <CR>

Where:

status 0 = Control-id/group-id not in use.
 1 = Control-id/group-id is in use.

Enabling/Disabling a Control

To enable or disable a named control (or control group), use the following AiF escape sequence:

ESC_11 {; enable} w control-id ESC

Where:

enable 1 = disable control.
 2* = enable control.

control-id Control id or group id.

Enabled controls will accept user input, disabled controls will not.

If you disable a control that currently has focus, or would have if the application were the active top level Window, then focus is shifted to the root.

Note: disabled controls are not greyed out - they simply will not accept any user input.

Showing/Hiding a Control

To show or hide a named control (group) on the screen, use the following AiF escape sequence:

ESC_12 {; show} w control-id ESC

Where:

show 1 = hide control.

2* = show control.

control-id Control id or group id.

If you hide a control that currently has focus, or would have if the application were the active top level Window, then focus is shifted to the root.

Destroying a Control

To destroy a named control, string list or control group, use the following AiF escape sequence:

ESC_10 {; delete} w control-id ESC

Where:

delete Use only if the id is that of a control group:
 1 = do not delete controls inside group
 2* = delete all controls in group.

control-id Control id, string list id or control group id.

Destroying a control will flush it from HostAccess's memory. The control is immediately removed from the screen.

If the specified control currently has focus, or would have focus if the application were the active top level Window, then focus is shifted to the root.

Deleting a control group will by default delete all the controls in that group. To retain the controls, set the **delete** parameter to 1.

Re-sizing/Moving a Control's Window

To re-size and/or move a control's window, use the following AiF escape sequence:

ESC_13 ; y ; x ; h ; wid w control-id ESC

Where:

y New y co-ordinate of top of control.
x New x co-ordinate of left of control.
h New height of control, in characters.
wid New width of control, in characters.
control-id Control id.

If **y** and **x** are set to (0,0), then the window is not moved.

Changing Control Colours

To change the foreground, background and grayed colours for a control, use the following AiF escape sequence:

```
ESC_14 {; fore} {; back} {; grayed} w control-id ESC\
```

Where:

- fore** Foreground colour, in range 1..16. (*=16).
- back** Background colour, in range 1..16. (*=1).
- grayed** Grayed colour, in range 1-16. (*=1).
Used by some controls when disabled.
- control-id** Control id.

How the colours are used depends on the control type and contents. Text labels for buttons are always shown in the foreground colour (unless the control is disabled).

Reporting Events

When an event is reported to the host, information about that event is sent in the following format:

```
<STX> WC<CR> id , event{, Argument} <CR>
```

Where:

- WC** Literal characters.
- id** Control id of control associated with the event or ? if no event available.
- event** Event number - see Event Numbers Defined below.
- Argument** Optional argument associated with event.

Event Numbers Defined

Currently defined event numbers are:

- 1 ENTER pressed.
- 2 ESCape pressed.
- 3 Button clicked.
- 4 Check box or radio button check state change.
Argument: 1 = button is now unchecked, 2 = button now checked.
- 5 Contents of edit box, or the contents of the edit box part of simple and dropdown combo boxes, have been changed by user.
Argument: edit box contents, escape sequence number 1, see Reading from an edit box on page 64, for the format.

- 6 List box selection change.
Argument: host string of newly selected item, or “?” if nothing selected.
- 7 List box double click.
Argument: host string of double clicked item.
- 8 FOCUS: sent whenever the user changes focus from one control to another.
Returns 4 parameters: old control (string label), event (1=ENTER, 3=CLICKED, 9=TABBED), New control (string label) and Amend flag (set to 2 if old control had changed since it gained focus, otherwise 1).
- 9 TAB: control has been tabbed from.
- 10 CLICKEDON: left mouse down event over control when a different control, or the root, had the focus, resulting in focus moving to the clicked on control.
Argument: id of control that has just lost the focus.
- 11 Secondary Window activate: the user is changing focus from a secondary window.
- 12 Secondary Window close: the user is trying to close a secondary window.
- 13 LISTBOX: tells host when a user scrolls off the end of a partially displayed string list.

Enabling Event Reporting

You can set or clear specific event reporting for named controls or control groups. For example, you could disable reporting for return keys pressed by the user.

To set/clear event reporting for controls/groups, use the following AiF escape sequence:

ESC_15 {; enable} ; event1 {checksum} w control-id ... ESC

Where:

enable	1 = disable events (discards outstanding stacked events). 2* = enable events. 3 = stack events.
event1 ...	Event number - see from page 22.
checksum	1* No checksumming 2 Use length checksum.
control-id	Is the control id or control group id.

Note: Destroying a control will flush outstanding events.

Examples

To disable button click reporting for button with id 'but1', use the following AiF escape sequence:

ESC_15 ; 1 ; 4 w but1 ESC

To enable enter key and button click reporting for button with id 'helpbut', use

`ESC_15 ; 2 ; 1 ; 3 w helpbut ESC\`

Requesting events for use with stacked events.

When you enable an event you can specify that the events are stacked. This means the events are not reported until your program is ready to receive the event. Then you can send an escape sequence to get the next event from the stack.

To request an event from the stacked event handling system, use the following AiF escape sequence:

`ESC_6 ; mode w {control_id} ESC\`

Where:

mode	1= Get next stacked event. Wait if no event is available.
	2= Get next stacked event. Return if no event is available.
	3= Get last reported stacked event.
	4= Flush event stack.
control_id	If mode is 1 or 2, control_id is optional and takes events only from that named control. Control_id is not relevant for modes 3 and 4.

Getting events

`ESC_6 ; {wait_code} w { control-id } ESC \`

Where:

wait_code	1 = get next event. Don't respond until an event is generated.
	2 = get next event, or return immediately.
	3 = get last event.
	4 = flush all events.

control-id1 ... Is the control id or control group id.

If `control_id` is specified, then the command applies only to that control. If it is not specified, then it applies to all controls.

This returns:

```
<STX> WC <CR> control event {argument} <CR>
```

Where:

control Control id or '?' if no event available.
event Event number.
argument Optional argument for event.

Setting Input Focus to a Named Control

To set the input focus to a named control, use the following AiF escape sequence:

```
ESC_16 w control-id ESC\
```

Where:

control-id Is the control id.

If the id given does not match a known control, or it is the string “root”, focus is returned to the background terminal characters.

Setting Input Focus to an Unknown Control

To set input focus to the next/previous control in the tabbing order, use the following AiF escape sequence:

```
ESC_17 ; direction w ESC\
```

Where:

direction 1 = set the input focus to the previous control.
 2* = set the input focus to the next enabled and visible control.

If there are no such controls, focus will be left with the root.

Using Control Groups

You can create control groups, holding several different controls, for ease of use. Once you have created groups of controls, you can use any of the generic control management facilities documented in this section on entire control groups (for example, showing/hiding controls).

To add or remove controls to/from a control group, use the following AiF escape sequence:

```
ESC_18 ; add w group-id ; control-id1 ... ESC\
```

Where:

add	1 = remove one or more controls from a control group 2* = add one or more controls to a control group.
group-id	Control group id.
control-id1	Individual control id(s). You have to define (create) each control id separately before using it with a control group.

Creating a Control Group

To create a new control group, add one or more controls to a group with the required id and the group will be automatically created.

Example

To create a control group named **buttons**, holding the controls **radio1**, **radio2** and **check1**, use the following AiF escape sequence:

```
ESC_18 ; 2 w buttons ; radio1 ; radio2 ; check1 ESC\
```

To delete the control **radio2** from that control group, use the following AiF escape sequence:

```
ESC_18 ; 1 w buttons ; radio2 ESC\
```

Returning an Alternate Message

To tell a control to return a different string to the host when the given event occurs for which reporting is enabled, use the following AiF escape sequence:

```
ESC_19 ; event ; class w control-id ; message ESC\
```

Where:

event	Event number, see page 22 for details.
class	1* = Send message back as a string (default behaviour). 2 = Treat message as the name of a macro file to execute. 3 = Send message back as a string and pass focus back to Terminal Window
control-id	Control id.

message Alternate message.

When an alternate message is set, it will be sent to the host unmodified. It will not have a CR sent after it.

For example, to cause a single character, 'X', to be sent to the host when the 'ed' control has been tabbed to (tab = event number 9), use:

```
ESC_19 ; 9 w ed ; X ESC\
```

Setting the Accelerator Character

To set the accelerator character for a named control, use the following AiF escape sequence:

```
ESC_20 w control-id ; accel ESC\
```

where **accel** is the accelerator character.

This allows the control to receive the focus when the user presses Alt plus the given key, but only when the control is capable of accepting the focus, and keystrokes are being processed by controls (if the root has the focus, they may not be). This does not change the visual appearance of the control at all. It is usual to indicate to the user what the accelerator key is by underlining it in the label nearest the control.

In the case of button controls (push, check and radio), there is no need to issue this escape, since the accelerator key will be set automatically by searching the label for the first and prefixed character.

Example

To allow **Alt/E** to be the accelerator key for control with id 'edit', use:

```
ESC_20 w edit ; E ESC\
```

Setting the Return Key Meaning

Normally, pressing a Return key reports a RETURN event to the host. To set the event returned (for a named control), use the following AiF escape sequence:

```
ESC_21 ; meaning w control-id ESC\
```

Where:

meaning	1 = set RETURN event returned to be a TAB. 2 = set RETURN event returned to be a RETURN (i.e. default return behaviour).
control-id	Control id.

Copy and Paste

The following AiF sequences enable you to copy a region of screen, including sculpting, and paste it into another specified region.

Copying an area of the screen

The following sequence copies a rectangular region of the screen, complete with sculpting, into a specified slot.

ESC_22 ; slot ; x ; y ; right ; bottom ; option w ESC

Where:

slot	Slot number. The minimum is 0, the maximum is 255. If the slot is already in use, it is overwritten with the new region. If the option specified is 1, then the region is also cleared of text and sculpting.
x	x co-ordinate of rectangle.
y	y co-ordinate of rectangle.
right	x right co-ordinate of rectangle.
bottom	y bottom co-ordinate of rectangle.
option	Choose one of the following options: 0 = Leave rectangle. 1 = Clear rectangle and save. 2 = Clear rectangle only.

Pasting a copied region of screen

The following sequence pastes a region of the screen copied into a slot using ESC_22 to either the position from which the region was saved, or at new co-ordinates if specified.

```
ESC_23 ; slot ; x ; y w ; esc\
```

Where:

- | | |
|-------------|--|
| slot | Slot number to restore. If an unused slot is specified, a journal message will be generated and the action ignored. |
| x | x co-ordinate where the saved region of screen will be pasted to. If x and y are left blank, the region will be pasted to the position from which the region was copied. |
| y | y co-ordinate where the saved region of screen will be pasted to. If x and y are left blank, the region will be pasted to the position from which the region was copied. |

Note: the region specified by x and y must be visible on the screen, offscreen regions will be ignored.

Clearing slots of copied screen regions

The following sequence clears all slots of screen regions copied using ESC_22.

```
ESC_24 w ESC\
```

Root Control Features

This section describes some escape sequences that may be used to manage the 'root' control. This is not really a control at all, but an interface to manipulate behavioural aspects of the underlying terminal character display area in so far as they relate to embedded controls.

To use these functions, you must first create the one and only 'root control'. Once created, subsequent attempts to create it are ignored.

Once created, you can use some of the standard control management escapes on it (such as the event management escape, if you're interested in, say, when the root is tabbed to).

Creating the Root Control

To create the one and only 'root' control, use the following AiF escape sequence:

```
ESC_25 w ESC\
```

It has the fixed id string 'root'.

Reading From the Root Control

To read a value from the 'root' control, use the following AiF escape sequence:

```
ESC_26 ; 1 w ESC\
```

A value will be returned to the host. The value returned is the 'tab in permitted' state of the root, sent in the following format:

```
<STX>value<CR>
```

Where:

```
Value  1 = tab-in is permitted.
         2 = tab-in is not permitted.
         ? = error detected.
```

Manipulating the Root Control

To disable or enable 'tab-in' to the root, use the following AiF escape sequence:

```
ESC_27 ; 1 ; {tabin} w ESC\
```

Where:

```
tabin  1 = disable 'tab-in' to the root.
         2* = enable 'tab-in'.
```

Naming a Base Control Group

To name a control group to which all subsequently created controls will be automatically added, use the following AiF escape sequence:

```
ESC_28 ; 1 w group ESC\
```

Where:

group The name of the control group. If this control group does not exist, it will be created.

Setting Default Foreground/Background Colours

To set the default foreground and background colours for use when subsequent controls are created, use the following AiF escape sequence:

```
ESC_28 ; 2 {; fore} {; back} {; grayed} w ESC\
```

Where:

fore	Foreground colour, in range 1-16.	*=16.
back	Background colour, in range 1-16.	*=1.
grayed	Grayed colour, in range 1-16.	*=5, for subsequently created controls.

Forcing Palette Reconstruction

To force palette reconstruction (back to the original default setup), use the following AiF escape sequence:

```
ESC_28 ; 3 w ESC\
```

This is useful in some situations after removing/adding 256 colour bitmaps. All controls showing bitmaps will be redrawn when this happens.

Secondary Windows

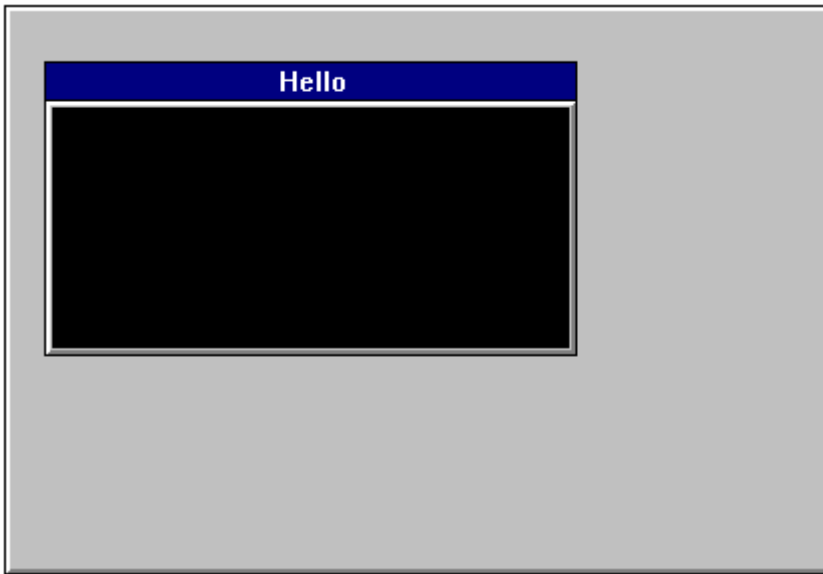
You create secondary windows inside your main terminal window. If the secondary window has been correctly defined and is activated, the user can perform normal Windows manipulation functions, such as:

- Entering data.
- Minimising/maximising the window.
- Re sizing and re-positioning the window.

This section describes how to create, destroy, activate, hide and show secondary windows.

Scaling

You can scale secondary windows, by defining the number of columns and rows in the window, then defining the window's actual size. By default, the scaling is 1, and the secondary window is created just large enough to hold the terminal window inside it.



This shows a secondary window, with top left corner at (3,3), holding a terminal window of width 40, height 10, and title "Hello".

Creating a Secondary Window

To create a secondary window, use the following AiF escape sequence:

```
ESC_29 ; 1 ; top ; left ; h1 ; wid1 {; wid2} {; h2} {; fg} {; bg} {; mod} {; bord} {; bar} {; min} {; max} {;
orig} {; size} {; page} w id ; title ESC\
```

Where:

Top Window top. Cell offset.

Note:

When **orig** = 1, this is the pixel row of the top/left of the window, i.e. one more than the number of pixels visible above the top of the sub window frame.

When **orig** = 2, this is one more than the number of pixels of the application window, including the window frame, visible above the top/left of the window frame.

When **orig** = 3, this is the row/column number measured in character cells within the base window of the top row/ first column of characters in the subwindow. The window border is drawn above this position.

Left Window left. Cell offset. See note in **top**.

h1 Number of columns in terminal window. If **h1** and **wid1** is smaller than **h2** and **wid2**, the font will be scaled down so the correct number of cells will still appear in the subwindow.

wid1 Number of rows in terminal window. If **h1** and **wid1** is smaller than **h2** and **wid2**, the font will be scaled down so the correct number of cells will still appear in the subwindow.

wid2 Window width - use to scale the window horizontally.

When **size** = 2 and **wid2** is smaller than **wid1**, the subwindow is initially drawn with a width of **wid2** character cells (measured by the cell size of the base window). If **size** = 1 or **wid2** is greater than **wid1**, **wid2** and **h2** will have no effect.

h2 Window height - use to scale the window vertically.

Fg Foreground colour of terminal window 1..16, *=16

bg Background colour of terminal window 1..16, *=1

mod Modality: 1 = modeless, 2* = modal.

bord Border type: 1 = none; 2* = thin, not resizeable, 3 = thick, resizeable.

bar Title bar type: 1 = none; 2* = normal.

If you have a title bar, a default thin border is used by default, although you can have a thick border using **bord**.

- min** 1*= do not show a minimise box.
 2 = show a minimise box.
 3 = do not show a minimise box but show close.
 4 = show minimise and close.
- max** 1*= do not show a maximise box.
 2 = show a maximise box.
 3 = do not show maximise box and create the window hidden.
 4 = Show maximise box and create the window hidden.
 The window will become modeless if windows are created hidden
- orig** Window position:
 1 = relative to the screen, in pixels.
 2 = relative to the application window, in pixels.
 3* = relative to the main terminal window, in character cells.
- size** Interpretation of secondary window size:
 1 = pixel size of whole window, including non-client parts.
 2* = size in cells of the displayed terminal window, to which the non-client parts are added.
- page** Specify the number of backpages where:
 0* =an active page and the number of backpages which have been specified in Configure, Edit from the HostAccess menu.
 1= an active page and no backpages.
 2 =an active page and 1 backpage.
 3 = an active page and 2 back pages and so on.
- Id** Control id of the secondary window.
- title** Title.

Example of how to create a secondary window.

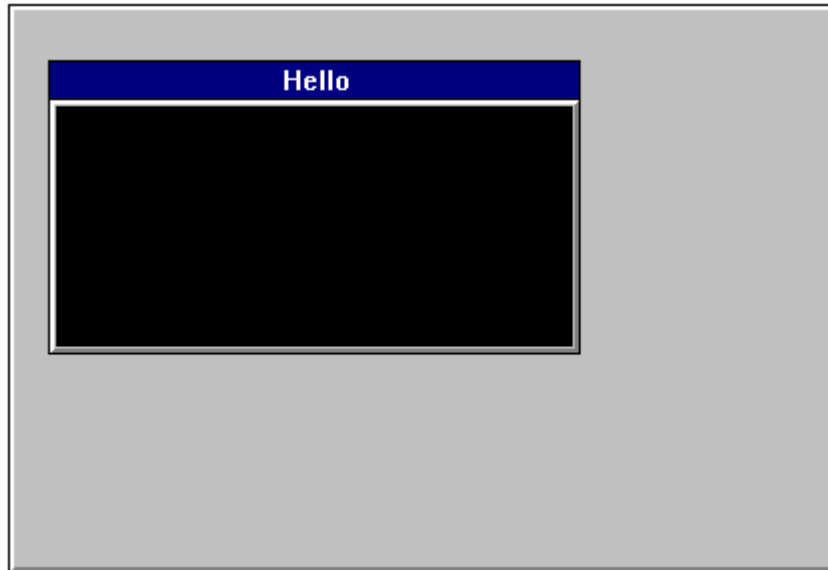
To create a secondary window at (3,3), holding a terminal window of size 40x10, use the following AiF escape sequence:

```
ESC_29 ; 1 ; 3 ; 3 ; 10 ; 40 w Help ; Hello ESC\
```

The Windows window will be made exactly the right size to hold the terminal window inside it.

The control id is 'Help'; the title will be 'Hello'

This will produce the following display on your terminal window:



Destroying a Secondary Window

To destroy secondary windows, use the following AiF escape sequence:

```
ESC_29 ; 2 w id ESC\
```

Where **id** is the window id.

Activating a Secondary Window

To activate secondary windows, use the following AiF escape sequence. This will bring the active window to the front.

```
ESC_29 ; 3 w id ESC\
```

Where **id** is the window id.

Setting Focus for Output in a Secondary Window

To set the focus for host output to a secondary window, use the following AiF escape sequence. This sequence is useful for controlling host output to different windows as the user may change the focus of the secondary window manually by clicking on the active window.

```
ESC_29 ; 3 ; 1 w id ESC\
```

Hiding/Showing a Secondary Window

To hide or show a secondary window, use the following AiF escape sequence:

```
ESC_33 ; 2 ; show w id ESC\
```

Where:

show	1 = Hidden. 2 = Minimised. 3* = Normal. 4 = Maximised.
id	Toolbox/window id.

Buttons

The following sections describe how to use AiF escape sequences to create and use:

- Text push button.
- Image push button.
- Images (treated as static buttons).
- Radio buttons.
- Check boxes.

When using these escape sequences, you can describe button images for a button in great detail.

Creating a Text Button

To create a pushbutton holding a text label, use the following AiF escape sequence:

```
ESC_30 ; y ; x ; h ; wid {; visible} {; enabled} {; font} w control-id ; label ESC\
```

Where:

y	y co-ordinate of top of button.
x	x co-ordinate of left of button.
h	Height of control, in character cell units.
wid	Width of control, in character cell units.
visible	1 = create hidden. 2* = create visible.
enabled	1 = initially disabled. 2* = initially enabled.
font	Selects button label font: 1* = Terminal. 2 = System. 3 = 10pt Helvetica. 4 = 8pt Helvetica. Styles 3 and 4 map on to the Helvetica fonts used in Borland-style bitmap pushbuttons and in dialog static text used by the application.
control-id	Control id – must be unique, and may not be “root”.
label	Button label.

Creating An Image Button

To create a push button holding a bitmap image and (optionally) a text label, use the following AiF escape sequence:

```
ESC_31 ; y ; x ; h ; wid {; visible} {; enabled} {; font} w control-id ; spec ESC\
```

Where:

y	y co-ordinate of top of button.
x	x co-ordinate of left of button.
h	Height of button, in rows.
wid	Width of button, in columns.
visible	1 = create hidden, 2 = create visible(*).
enabled	1 = initially disabled, 2 = initially enabled(*).
font	Selects button label font: 1* = Terminal. 2 = System. 3 = 10pt Helvetica. 4 = 8pt Helvetica. Styles 3 and 4 map on to the Helvetica fonts used in Borland-style bitmap pushbuttons and in dialog static text used by the application.
control-id	Control id.
spec	See Appendix A, Describing Images for details.

Image Specification for Pushbuttons

This powerful feature allows you to create pushbuttons holding:

- Bitmap images (.BMP files).
- Icon images (.ICO files).
- Bitmaps or icons in resource files (.DLL or .EXE files).

Example

To create a pushbutton called **help**, displayed at (10,10), with height 5 and width 10, using the bitmap image held in the file `c:\pictures\question.bmp`, use the following AiF escape sequence:

```
ESC_31 ; 10 ; 10 ; 5 ; 10 w help ; file=c:\pictures\question.bmp ESC\
```

See Appendix A, Describing Images for details.

Displaying an Image

To display an image on the screen, you can create a disabled push button, with an image defined using an image specification string. This allows a simple way of displaying icons or bitmap images.

To display an image, use the following AiF escape sequence:

```
ESC_32 ; y ; x ; h ; wid {; visible} w control-id ; spec ESC\
```

Where:

y	y co-ordinate of top of button.
x	x co-ordinate of left of button.
h	Height of control, in character cell units.
wid	Width of control, in character cell units.
visible	1 = create hidden. 2* = create visible.
control-id	Control id.
spec	See Appendix A, Describing Images for details.

Note: the label font is set to the terminal font. This does not usually matter since disabled buttons normally just hold an image.

Example

To display an image called **asterisk**, displayed at (10,10), with height 5 and width 10, using the bitmap image held in the file **star.bmp**, use the following AiF escape sequence:

```
ESC_32 ; 10 ; 10 ; 5 ; 10 w asterisk ; file=star ESC\
```

Creating a Check box

To create a check box, use the following AiF escape sequence:

```
ESC_34 ; y ; x ; h ; wid {; visible} {; enabled} {; font} {; left} {; check} w control-id ; label ESC\
```

Where:

y	y co-ordinate of top of check box.
x	x co-ordinate of left of check box.
h	Height of check box, in rows.
wid	Width of check box, in columns.
visible	1 = create hidden. 2* = create visible.
enabled	1 = initially disabled. 2* = initially enabled.
font	Selects font: 1* = Terminal. 2 = System. 3 = 10pt Helvetica. 4 = 8pt Helvetica.
left	1 = text on left. 2* = text on right.
check	1* = initially unchecked. 2 = initially checked.
control-id	Control id - must be unique, and may not be "root".
label	Text label.

The event reporting mask is initially set to all bits clear.

Creating a Radio Button

To create a radio button, use the following AiF escape sequence:

```
ESC_35 ; y ; x ; h ; wid {; vis} {; en} {; font} {; left} {; check} w r-id {; label} {; g-id} ESC\
```

Where:

y	y co-ordinate of top of radio button.
x	x co-ordinate of left of radio button.
h	Height of radio button, in rows.
wid	Width of radio button, in columns.
vis	1 = create hidden. 2* = create visible.
en	1 = initially disabled. 2* = initially enabled.
font	Selects font: 1* = Terminal. 2 = System. 3 = 10pt Helvetica. 4 = 8pt Helvetica.
left	1 = text on left, 2* = text on right.
check	1* = initially unchecked, 2 = initially checked.
r-id	Radio button control id - must be unique, and may not be “root”.
label	Text label for button - optional.
g-id	Control group id, if relevant - optional.

The event reporting mask is initially set to all bits clear.

Using Radio Buttons in Groups

If you give a control group id, the button control is automatically added to that group. The group is created if it does not exist.

When the first radio button control is added to a radio button group, it is forced to be checked, even if the host has not asked for it. When subsequent radio buttons are added to a radio button group, if an initially checked button is added, the check is removed from the previously checked button in the group.

These rules ensure that exactly one radio button in a group will be initially checked. It also means that when creating the controls, if they are created as visible, and the initially checked button is not going to be the first, the user will momentarily see the check on the first button. To avoid this, create radio buttons initially hidden, and then show them all at once.

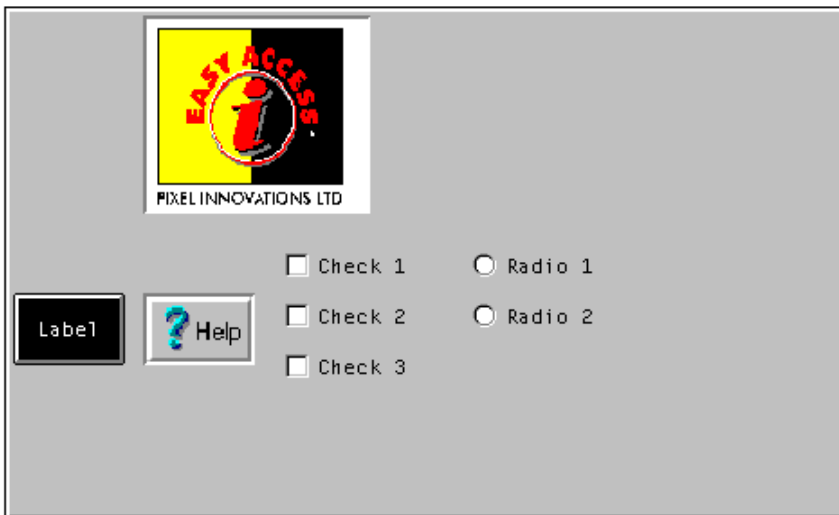
Example: Using Buttons

This example displays the following on your terminal window:

- A 3x8 push button called **text**, at (12,1), labelled "Label".
- A 3x8 image button called **help**, at (12,10), using the image held in the file `c:\bitmaps\flhelp.bmp`.
- A 8x16 image called **logo**, at (1,10), using the image `c:\bitmaps\easyacc.bmp`.
- Three check boxes: **check1**, **check2** and **check3**.
- Two radio button: **radio1** and **radio2**.

using the following AiF escape sequences:

```
ESC_30 ; 12 ; 1 ; 3 ; 8 w text ; Label ESC\
ESC_31 ; 12 ; 10 ; 3 ; 8 w help ; file=c:\bitmaps\flhelp.bmp ESC\
ESC_32 ; 1 ; 10 ; 8 ; 16 w logo ; file=c:\bitmaps\easy-acc.bmp ESC\
ESC_34 ; 10 ; 20 ; 2 ; 10 ;;; 2 w check1 ; Check 1 ESC\
ESC_34 ; 12 ; 20 ; 2 ; 10 ;;; 2 w check2 ; Check 2 ESC\
ESC_34 ; 14 ; 20 ; 2 ; 10 w check3 ; Check 3 ESC\
ESC_35 ; 10 ; 33 ; 2 ; 10 w radio1 ; Radio 1 ESC\
ESC_35 ; 12 ; 33 ; 2 ; 10 ;;; 2 w radio2 ; Radio 2 ESC\
```



Reading a Button

You can read when check boxes or radio buttons have been checked, as described in the following sections.

In all cases, a value will be sent to the host. This is formatted as:

```
<STX><value> <CR>
```

If an error is detected in the arguments, the value returned is a single question mark (STX ? CR).

Reading a Button's Check State

To read the check state of a button, use the following AiF escape sequence:

```
ESC_36 ; 1 w control-id ESC\
```

Where:

control-id Is the id of the button.

All buttons have a check state, but it is only meaningful to read the state of check boxes and radio buttons.

Reading Which Button is Checked

To read the id of a radio button in the group that is currently checked, use the following AiF escape sequence:

```
ESC_36 ; 2 w control-id ESC\
```

Where:

control-id Is the id of the button group.

Setting/Clearing a Button

To set or clear a given radio button or check box, use the following AiF escape sequence:

```
ESC_37 ; 1 ; change w control-id ESC\
```

Where:

change 1 = clear (uncheck) the radio button or check box.
2* = set (check) the radio button or check box.

control-id Is the id of the button group.

When doing this to a radio button that is part of a radio button group, the button is always checked; and the previously checked button in the group (there must be one) is always unchecked.

String Lists

String lists are used in conjunction with list boxes and combo boxes. String lists contain the entries used to populate these boxes. String lists are created and managed quite separately to the list/combo boxes which use them. You can therefore use a single string list in multiple boxes, and create and destroy boxes without destroying the underlying data.

Creating String Lists

To create string lists, either download the strings from the host, or read them in from a PC file. The second form is better suited to longer lists. Although there is no inbuilt limit to the number of items in such a list, they are not intended for very large lists, because:

- A list box control cannot contain more than 64k of text, for example, if the average string length is 90 bytes, a list box will not hold more than approximately 700 items.
- String lists are held in memory at all times.

- The time required to create large lists will be unacceptable to users.
- The time required to populate list/combo boxes with large lists will be unacceptable to users.

A realistic limit is a few hundred items.

Format of String Lists

In its simplest form, a string list has an **id** (a name), and an ordered sequence of strings. The order defines the default order in which the strings are displayed in a list or combo box (although this can be changed when the boxes are actually created).

String lists may also store a second, hidden, string for each item. This string is not displayed to the user, but can be used by the host application as an alternative way for list items to be specified in messages exchanged between HostAccess and the host. See pages 45 and 46 for examples.

Manipulating String Lists

To create, add entries to and remove entries from, string lists, use the following AiF escape sequence:

```
ESC_40 {; add} w string-id ; text ; entry1... ESC\
```

Where:

add	1* = add strings to string list. 2 = remove strings from string list.
string-id	String list id.
text	The display text of the string list entry before which the new strings are to be inserted. Ignored if removing entries.
entry1 ...	1st and subsequent entries to be added/removed.

The entries in the list contain a 'display' part, and optionally, a 'hidden' part. If present, the hidden part is separated from the display part by a comma (so you cannot have a comma in the display part). If the hidden part is not given, a default hidden value will be automatically created if it is ever needed- this will be a string representation of the position of the entry in the string list (starting from 1; '1', '2', '3' etc).

List entries to be added/removed are given directly or indirectly. When given directly, a string parameter specifies the list entry in the format:

```
<display-part> <, hidden-part>
```

If a string parameter starts with an '@' character, it is treated as an indirect entry. The '@' is stripped off, and the remainder treated as a PC file name. The file contains a list of entries in the above format. It is possible to mix the direct and indirect forms in a single escape sequence.

Note: the comma and '@' characters cannot normally be used in display strings because of their special significance in the above formats. However they can be changed, see page 48 for details.

When adding strings, the second string parameter contains the display text of the existing string list entry before which the new entries are to be inserted. If missing, the new entries are added to the end of the list.

When removing entries, the hidden parts of entries are ignored.

Example of a string list

Consider a host application that needs to get the user to select a personnel record from a database. Each record includes the person's name. Each record has a record number. The host application wants to use a drop-down list style combo box (one in which the user cannot type an entry, but has

to select from the list) to get the name. The host application is not really interested in the text of the name, but the record number it relates to.

This is more suited to a string list with hidden strings. The 'display' strings are the names of the people, the hidden strings are the associated record numbers. The host creates such a string list, then associates it with a 'dropdown combo' style box. The host also specifies that it wants to use the hidden strings when exchanging information with HostAccess about the selected list items. HostAccess will then send back the record number of the selected item, which the host application can use directly.

Create a list of people, with hidden strings (record numbers in some database). The bulk of the list is created from a PC file called **people.lst**. To this are added 2 people given directly. The list will be called 'people' and will eventually contain the following entries, in the order given:

Display text	Hidden text	Source
D. Bailey	173	People.lst
M. Woolley	174	People.lst
G. Baker	190	People.lst
F. Carden	191	People.lst
A.Hedgecock	160	Direct from host
P.Hall	143	Direct from host

ESC_40 w people ;; @people.lst ; A.Hedgecock, 160 ; P.Hall, 143 ESC

people.lst is a standard DOS file with <CR><LF> characters separating each line of text. The file name could also include the full directory path, for example **c:\windows\data\people.lst**. By default, the path is your HostAccess directory. In this example, **people.lst** looks like this:

```
D.Bailey, 173
M.Woolley, 174
G.Baker, 190
F.Carden, 191
```

Example 2 - String lists

The host application has a screen on which one of the pieces of information the user has to enter is a city name. The host application designer chooses to do this with a simple combo box, which has a list of common cities, but will also let the user type in a city that's not on the list. All the host application wants to get from the user is the text of the city name.

This is best suited to the simple form of string list, without use of hidden strings. The host downloads the list of cities in a string list, then creates a 'simple combo' style box. When extracting the selected city, or the name the user entered, HostAccess sends the relevant text to the host.

Example

Create a list containing the following cities: Birmingham, Bristol, Coventry, Leeds, London, Manchester, and York, called 'cities'.

```
ESC_40 w cities ;; Birmingham ; Bristol ; Coventry ; Leeds ; London ; Manchester ; York ESC\
```

Reading From a String List

In all cases, a value will be returned to the host. This is formatted as:

```
<STX> value <CR>
```

If an error is detected in the arguments, the value returned is a single question mark (STX ? CR).

Reading String List Size

To return the number of items in a string list, use the following AiF escape sequence:

```
ESC_41 ; 1 w control-id ESC\
```

Reading Selected Display Text

To return the display text for the selected list item, use the following AiF escape sequence:

```
ESC_41 ; 2 ; item w control-id ESC\
```

Where **item** is the number of the relevant item (starting from 1).

Reading Selected Hidden Text

To return the hidden text for the selected list item, use the following AiF escape sequence:

```
ESC_41 ; 3 ; item w control-id ESC\
```

Where **item** is the number of the relevant item (starting from 1).

Clearing a String List

To delete all entries in a string list, use the following AiF escape sequence:

```
ESC_42 ; 1 w control-id ESC\
```

Where **control-id** is the control id for the string list.

Setting Special Characters

To set hidden text separator and indirect entry characters, use the following AiF escape sequence:

```
ESC_42 ; 2 w control-id ; string ESC\
```

Where **string** is a 2-character string holding these characters in order.

For example, to set the default special characters (, @) in the string list named **string1**, use the following AiF escape sequence:

```
ESC_42 ; 2 w string1 ; ,@ ESC\
```

Note:

To include semicolons within strings, put a pipe character in front of the semicolon, e.g.

```
ESC_40 w TEST; ; This is a semicolon |; in the text ; this is item 2 in the list ESC\
```


Combo Boxes

The following sections describe how to create, read and manipulate combo boxes: A combo box can combine an edit box with a drop-down string list (see example on page 50).

Creating a Combo Box

To create a combo box, use the following AiF escape sequence:

```
ESC_45 ; y ; x ; h ; wid { ; vis } { ; en } { ; font } { ; box } { ; sort } { ; bar } { ; msg } { ; auto } { ; border } w c-id
{ ; str-id } { ; sel } ESC\
```

Where:

y	y co-ordinate of top of box.
x	x co-ordinate of left of box.
h	Height of box, in rows.
wid	Width of box, in columns.
vis	1 = create hidden, 2* = create visible(*).
en	1 = initially disabled, 2* = initially enabled.
font	Text font: 1* = Terminal. 2 = System. 3 = 10pt Helvetica. 4 = 8pt Helvetica.
box	2 = simple combo box. 3 = dropdown combo box. 4 = dropdown list combo box. 5 = simple combo box borderless. 6 = dropdown combo box, borderless. 7 = drop-down list combo box, borderless.
sort	1 = unsorted. The list items appear in the same order as in the string list. 2* = sorted. The list items will appear in alphabetical order.
bar	1 = no scroll bar, even if items too wide for box, 2* = use scroll bar, if needed.
msg	1* = messages sent between host and HostAccess will use display text. 2 = messages sent between host and HostAccess will use hidden text. msg selects whether the host wishes to use the display text or hidden text of string list entries when communicating with HostAccess and affects messages sent in both directions. Str-id and sel will be interpreted as display or hidden text depending on this value. It also affects subsequent event reporting (selection change and double click events), and the way items are specified and transmitted in other escape sequences.

- auto** 1 = no automatic horizontal scroll in edit box of combo box.
2* = automatic horizontal scroll in edit box of combo box.
- border** 1 = no border. The control uses the full depth of the control's rectangle, probably displaying a partial item at the bottom.
2* = normal border, only show integral no. of items.
- c-id** Control id of combo box.
- str-id** String list id, optional. If not given, list will initially be empty.
- sel** Display/hidden text of item to be initially selected - optional. If not given, the first displayed entry in the list/combo box will be initially selected.

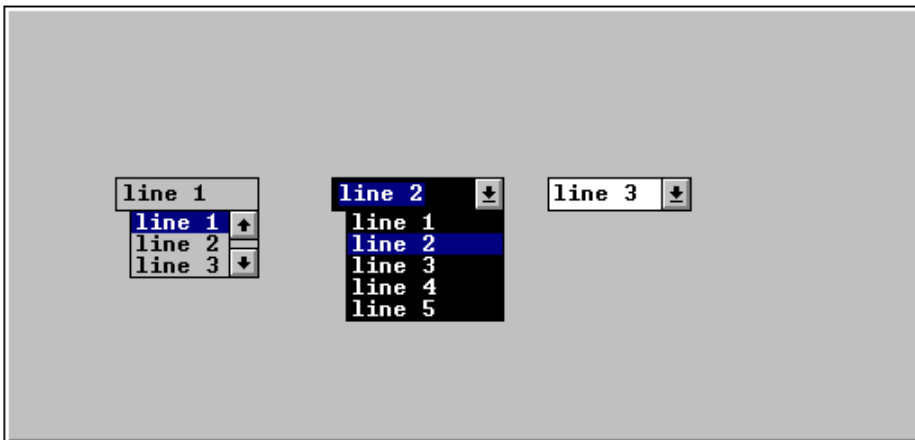
Combo Box example

This example creates a string list with control id **str-list**. It then creates:

- A 5x10 simple combo box (**combo1**) at (10,10).
- A 7x12 dropdown combo box (**combo2**) at (10,25).
- Aa 5x10 drop-down list combo box (**combo3**) at (10,40).

All the boxes use **str-list** for their contents. Note that **combo1** has changed background colour. See page 22 for details of this feature.

```
ESC_40 w str-list ;; line 1 ; line 2 ; line 3 ; line 4 ; line 5 ESC\
ESC_45 ; 10 ; 10 ; 5 ; 10 ;;; 2 w combo1 ; str-list ESC\
ESC_14 ; 1 ; 8 w combo1 ESC\
ESC_45 ; 10 ; 25 ; 7 ; 12 ;;; 3 w combo2 ; str-list ESC\
ESC_45 ; 10 ; 40 ; 5 ; 10 ;;; 4 w combo3 ; str-list ESC\
```



Reading Combo Boxes

You can read from a combo box, using the escape sequence described in the following sections to return a value to the host.

In all cases, a value will be sent to the host. This is formatted as:

```
<STX><value> <CR>
```

If an error is detected in the arguments, the value returned is a single question mark (STX ? CR).

Reading the Current Item in a Combo Box

To return the contents of the currently selected item, use the following AiF escape sequence:

```
ESC_46 ; 1 w control-id ESC\
```

The contents consists of either the display or hidden text for the selected item, depending on the value of the `msg` parameter when the combo box was created. See page 49.

If hidden text is returned, but was not defined for the selected entry, the position of the item in the string list is returned. This may not be the position of the selected item as displayed. If the box was created with alphabetic sorting turned on, the order of presentation in the combo box is quite separate from the order in the string list. The value that is returned will always be the order in the string list.

If no item is selected, the value returned is a single question mark (STX ? CR).

Reading if a Combo box is Visible

To return whether or not the list portion of a combo box is 'dropped down' (i.e. visible), use the following AiF escape sequence:

```
ESC_46 ; 3 w control-id ESC\
```

The value returned is 1 if it is not visible, 2 if it is.

Reading Changes to Combo Boxes

To read if the contents of the box have been changed by the user, use the following AiF escape sequence:

```
ESC_46 ; 4 w control-id ESC\
```

The value returned is 1 if unchanged, 2 if changed.

This applies to simple or dropdown combo box styles only (not dropdown list).

Reading the Contents of a Box

To read the contents of a box, use the following AiF escape sequence:

```
ESC_46 ; 5 { length} w control-id ESC\
```

Where:

Length The maximum length that is to be returned. (*=80).

This applies to simple or dropdown combo box styles only (not dropdown list).

The contents of the box are returned.

Reading Selected Characters

To return edit box selection indication (telling the host which characters are selected), use the following AiF escape sequence:

ESC_46 ; 6 w control-id ESC

The value returned is two comma-separated integers **n,w**, where:

n The number of the first character in the selection (starting from 1).

w The number of selected characters.

If nothing is selected, the return value is '1,0'.

This applies to simple or dropdown combo box styles only (not dropdown list).

Setting the Current Item in a Combo Box

To set an item to be selected, use the following AiF escape sequence:

ESC_47 ; 1 w control-id ; item ESC

Where:

item Display/hidden text of the required items.

The items are specified as display/hidden text of the required items, depending on the value passed in the **msg** parameter when the combo box was created.

See page 49 for details.

Changing the String List to be Displayed in a Box

To change the string list that is to be displayed in the box, use the following AiF escape sequence:

ESC_47 ; 2 w control-id ; string-id ESC

Where **string-id** is the (optional) string list id. If omitted, the box becomes empty.

Limiting Text in Combo Boxes

To limit the amount of text that may be entered, for simple or dropdown combo box styles only, use the following AiF escape sequence:

ESC_47 ; 3 ; limit w control-id ESC

Where **limit** is the limit.

Setting the Edit Box Selection Range

To set the edit box selection range, for simple or dropdown combo box styles only, use the following AiF escape sequence:

ESC_47 ; 4 ; start ; length w control-id ESC

Where:

Start The position (starting from 1) of the first character to be selected.

Length The number of characters that are to be selected.

Hiding and Showing Combo Boxes

To drop-down (show) or close up (hide) the list box part of the combo box, for simple or dropdown combo box styles only, use the following AiF escape sequence:

ESC_47 ; 5 {; show} w control-id ESC

Where:

show 1 = hide.
 2* = show.

Using the Clipboard (combo box styles)

For simple or dropdown combo box styles only, you can use the clipboard facilities as follows:

To cut the selection in the box to the clipboard, use the following AiF escape sequence:

ESC_47 ; 6 w control-id ESC

To copy the selection in the box to the clipboard, use the following AiF escape sequence:

ESC_47 ; 7 w control-id ESC

To paste the clipboard contents into the box at the current insertion point, use the following AiF escape sequence:

ESC_47 ; 8 w control-id ESC

This is ignored if the clipboard does not contain text.

To clear the current selection in the box (deleting it without placing it in the clipboard.), use the following AiF escape sequence:

ESC_47 ; 9 w control-id ESC

List Boxes

The following sections describe how to create both ordinary and incremental list boxes, how to read from and manipulate a list box.

Creating list boxes

To create a list box, use the following AiF escape sequence:

```
ESC_45 ; y ; x ; h ; wid { ; vis } { ; en } { ; font } { ; box } { ; sort } { ; bar } { ; msg } { ; auto } { ; border }
{ ; size } { ; style } w id { ; str-id } { ; sel } { ; top } ESC\
```

Where:

y	y co-ordinate of top of box.
x	x co-ordinate of left of box.
h	Height of box, in rows.
wid	Width of box, in columns.
vis	1 = create hidden, 2* = create visible.
en	1 = initially disabled, 2* = initially enabled.
font	Text font: 1* = terminal, 2 = system, 3 = 10pt Helvetica, 4 = 8pt Helvetica.
box	1* = list box, possibly incremental, see page 56 for details. 8 = tabular list box. This is a list box supporting tab characters, allowing you to input data in columns, see page 57 for an example.
sort	1 = unsorted. The list items appear in the same order as in the string list. 2* = sorted. The list items will appear in alphabetical order.
bar	1 = no horizontal scroll bar, even if items too wide for box. 2* = use horizontal scroll bar, if items too wide for box.
msg	1* = messages sent between host and HostAccess will use display text. 2 = messages sent between host and HostAccess will use hidden text. msg selects whether the host wishes to use display or hidden text.
auto	1 = no auto horizontal scroll in edit box of combo box. 2* = auto horizontal scroll in edit box of combo box.

border	1 = no border, list will try to use whole of control rectangle. 2* = normal border, only show integral no. of items. 3 = 3D Sculpted list type.
size	Sets the number of elements the list box will hold. For use with incremental list boxes. This must be at least one more than the number of elements.
style	0 = * standard incremental. Registers an event if a user pages off the bottom of the list box. 1 = extended incremental style. Registers events if the following occurs: 1: paging off the bottom of the list box. -1: Paging off the top of the list box.
id	Control id of list box.
str-id	String list id, optional. If not given, list will initially be empty.
sel	Display/hidden text of item to be initially selected - optional. If not given, the first displayed entry in the list/combo box will be initially selected.
top	Display/hidden text of item to be initially shown at the top of the box – optional. By default, the initially selected item is placed top most if possible.

Incremental List Boxes

You can use this feature to create list boxes with room for many entries, and create a corresponding string list with only a few strings.

This feature is useful if data transmission is slow, allowing you to update the list box incrementally as the user scrolls downwards.

You can get HostAccess to send notification messages to the host, whenever the user scrolls off the bottom of the visible strings, and so reveal an undefined entry. To do this, you need to enable event number 13.

This notification takes the format:

<STX>13,<element number>,<number of elements><CR>

If the host defines a string list which is larger than the total elements set then the total elements becomes the number of strings in the string list.

When the notification is received, the host should respond by adding the required string to the end of the string list associated with the list box.

See page 23 for a description of enabling event numbers.

See page 45 for a description of adding a string to a string list.

Note: sorting is automatically disabled for incremental list boxes.

Example: Incremental List Boxes

The host creates a list box with 100 entries, containing the entries in a string list named **Fill-up**, which contains only 10 strings.

The list box will display the 10 given strings and the remaining 90 will be empty.

The user may scroll down to reveal element 11 which is not available. HostAccess then sends

```
<STX>13,11,1<CR>
```

to the host. The host will then respond by adding a string (say, "Line 11") to the end of the string list (after "Line 10") associated with the list box, using the following AiF sequence:

```
ESC_40 w fill-up ; Line 10 ; Line 11 ESC\
```

The text "Line 11" will then be displayed in the list box.

Example 2: Incremental List Box

The following example creates a string list named **str-list**, containing the data described, then creates a simple list box, and a tabular list box, then sets the tab stops. The "→" symbol is used here to denote a tab character.

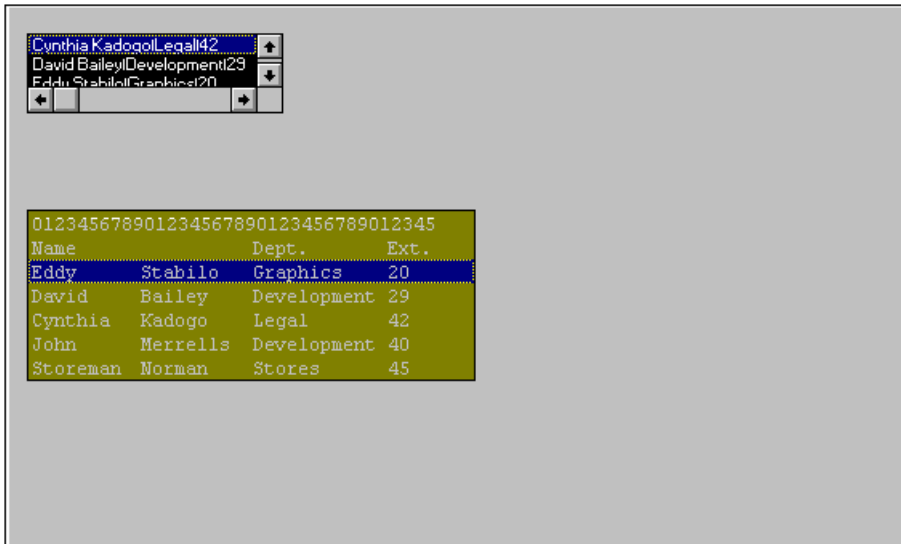
```
ESC_40 ; 1w str-list ;; 012345678901234567890123456789012345 ; Name→→Dept.→Ext. ;
Eddy→Stabilo→→Graphics→20 ; David→Bailey→Development→29 ;
Cynthia→Kadogo→Legal→42 ; John→Merrells→Development→40 ;
Storeman→Norman→Stores→45 ESC\
```

```
ESC_45 ; 1 ; 1 ; 4 ; 20 ;; ; 3 ; 1 w listbox1 ; str-list ESC\
```

```
ESC_45 ; 10 ; 1 ; 7 ; 40 ;;; ; 8 ; 1w listbox2 ; str-list ESC\
```

```
ESC_47 ; 11 ; 10 ; 20 ; 32 ; 40 ; 50 w listbox2 ESC\
```

```
ESC_14 ; 16 ; 4 w listbox2 ESC\
```



Note that the first box has a different font (8 point Helvetica) and background colour.

Reading from a List Box

You can read from a list box, using the AiF escape sequence described in the following sections to return a value to the host. This return value is formatted as:

`<STX> value <CR>`

If an error is detected in the arguments, the value returned is a single question mark (STX ? CR).

Reading the Current Item

To return the display or hidden text of the currently selected item, use the following AiF escape sequence:

`ESC_46 ; 1 w control-id ESC\`

The text returned depends on the value passed in the `msg` parameter when the list box was created. See page 52 for details.

If no hidden text was specified for the selected entry, the position of the item in the string list is returned.

Note: this may not be the position of the selected item as displayed. If the box was created with alphabetic sorting turned on, the order of presentation in the list box is quite separate from the order in the string list. The value that is returned will always be the order in the string list.

If no item is selected, '?' is returned.

Reading the Top Item

To return the display or hidden text of the top visible item, use the following AiF escape sequence:

`ESC_46 ; 2 w control-id ESC\`

Reading Total Size

To return the total number of elements defined, use the following AiF escape sequence:

```
ESC_46 ; 7 w control-id ESC\
```

Manipulating a List Box

The following areas are described:

- Setting the current item.
- Changing the string list to be displayed.
- Converting to incremental style.

Setting the Current Item

To set an item to be selected, and optionally also to be the topmost visible item, use the following AiF escape sequence:

```
ESC_47 ; 1 w control-id ; item ESC\
```

Where:

item Display/hidden text of the required items.

The items are specified as display/hidden text of the required items, depending on the value passed in the **msg** parameter when the list box was created

See page 55 for details.

Changing the String List to be Displayed

To change the string list that is to be displayed in the box, use the following AiF escape sequence:

```
ESC_47 ; 2 w control-id ; string-id ESC\
```

Where **string-id** is the (optional) string list id. If omitted, the box becomes empty.

Converting to Incremental Style

To convert a non-incremental style list box into an incremental style list box, and set the total number of elements, use the following AiF escape sequence:

```
ESC_47 ; 10 { elements } w control-id ESC\
```

Where:

elements The total number of elements in the box.

If the list box is already of incremental style then the total number of elements will be set to the new value. The total number of elements is always greater than or equal to the number of strings in the associated string list. This will fail if the list box is sorted.

Setting Tabs

If you have created a tabular list box, you can set one or more tab stops for that box. To set tab stops, use the following AiF escape sequence:

```
ESC_47 ; 11 ; width1... w control-id ESC\
```

Where:

width1 ... Width of tab stops (in characters). To set all the tab stops to be the same width send only one value. To set a list of tab stops send a tab position value for each tab stop.

control-id Control id of the tabular list box.

By default the tab stops are set to be half a system character width. The values must be sorted in increasing order.

Note: Back-tabs are not supported.

Edit Boxes

The following sections describe how to create, read and manipulate an edit box.

Creating an Edit Box

To create an edit box, use the following AiF escape sequence:

```
ESC_50 ; y ; x ; h ; wid {; vis} {; en} {; font} {; display} {; auto} {; acc} {; focus} {; edit} {; border}
{; scroll} ins/ovr w control-id {; contents} ESC\
```

Where:

y	y co-ordinate of top of box.
x	x co-ordinate of left of box.
h	Height of box, in rows.
wid	Width of box, in columns.
vis	1 = create hidden, 2* = create visible.
en	1 = initially disabled, 2* = initially enabled.
font	Selects font: 1* = terminal, 2 = system, 3 = 10pt Helvetica, 4 = 8pt Helvetica.
display	1* = display contents normally. 2 = force contents to upper case. 3 = force contents to lower case. 4 = 'password' - contents displayed as asterisks. (The password character may be switched from asterisk to something else).
auto	1 = do not automatically (horizontally) scroll the box. 2* = automatically (horizontally) scroll the box.
acc	1* = read/write access. 2 = read only access – user cannot change contents.
focus	1* = initially, contents not selected when box receives focus. 2 = initially, contents selected when box receives focus.
edit	1 = single line edit (* if height is 1). 2 = multi-line edit (* if height >1). 3 = multi-line edit with auto vertical scrolling.
border	1 = no border. The edit box height is exactly the multiple of character cells given. 2* = border. The box extends 4 pixels above and below the normal box rectangle. This means that you cannot have two consecutive edit boxes on two consecutive lines. 3 = 3D border.

Reading From an Edit Box

To read a value from an edit box, use the AiF escape sequences as described in the following sections.

In all cases, a value will be sent to the host. This return value is formatted as:

<STX><value> <CR>

If an error is detected in the arguments, the value returned is a single question mark (STX ? CR).

Reading a Line in an Edit Box

To read the contents of a given line in the box, use the following AiF escape sequence:

ESC_51 ; 1 ; max-len ; line w control-id ESC

Where:

max-len Is the maximum length of the text returned. (*=80).

Remember to set **max-len** when reading a multiline control, since their contents will often exceed 80 characters.

line Is the number (starting from 1) of the line you want. (*=all lines).

For single line edits, **line** is ignored, and the whole contents of the single line are returned, followed by CR.

For multi line edits, If **line** is given (and is greater than zero), then the contents of the specified line, only, are returned in the same format as for a single line edit.

If **line** is not given (or is given as 0), all lines in the edit box will be returned. Each line will be separated from the next by CR. Preceding the lines is the line count.

For example, if a multi-line edit contains 2 lines 'hello' and 'there', the reply would look like this:

2,hello<cr>there<cr>

For example, to return the contents of line 3 of edit box 'ed', use:

ESC_51 ; 1 ;; 3 w ed ESC

80 characters at most will be returned.

Examining Details of Lines

To read the number of lines in a box (always 1 for single line edit), use the following AiF escape sequence:

```
ESC_51 ; 2 ; w control-id ESC\
```

To read the current length of a given line, use the following AiF escape sequence:

```
ESC_51 ; 3 {; line} w control-id ESC\
```

Where:

line The relevant line number (*=1).

To read the line number of the first visible line in the box (for multi-line edits only), use the following AiF escape sequence:

```
ESC_51 ; 4 ; w control-id ESC\
```

Detecting Changes in an Edit Box

To return a flag that says if the edit box contents have been changed by the user.

```
ESC_51 ; 5 {; reset} w control-id ESC\
```

Where:

reset 0* = do not reset changed flag.
 1 = reset changed flag.

The value returned is '1' if no change, or '2' if changed.

Telling the Host Which Characters are Selected

To return a selection indication - telling the host which characters are selected (for single line edits), use the following AiF escape sequence:

```
ESC_51 ; 6 ; line w control-id ESC\
```

The value returned is two comma-separated integers **n,w**, where:

n The number of the first character in the selection (starting from 1).
w The number of selected characters.

If nothing is selected, the return value is '1,0'.

Manipulating an Edit Box

To manipulate an edit box, use the following AiF escape sequence features.

- Setting contents in an edit box.
- Limiting text entered.
- Scrolling.
- Changing 'Password' Character.
- Setting Selection Range.
- Using the Clipboard.
- Initialising a Multi-line Edit Box.

There is no reply to this escape sequence.

Setting Contents in an Edit Box

To set the contents of an edit box, use the following AiF escape sequence:

```
ESC_52 ; 1 w control-id ; contents ESC\
```

Where:

contents The new contents of the edit box.

Limiting Text Entered

To limit the amount of text that may be entered into the box, use the following AiF escape sequence:

```
ESC_52 ; 2 ; limit w control-id ESC\
```

Where:

limit The maximum number of characters. This is a total limit, not just the limit on a single line (multi-line edit box users take note).

Scrolling

To scroll an edit box so that that the given line number is the first visible line (for multi line edits only), use the following AiF escape sequence:

```
ESC_52 ; 3; line w control-id ESC\
```

Where

line The relevant line number.

Changing “Password” Character

To change the “password” character, use the following AiF escape sequence:

```
ESC_52 ; 4 ; w control-id ; char ESC\
```

Where:

char The password character.

Setting Selection Range

To set the selection range (for single line edit boxes only), use the following AiF escape sequence:

```
ESC_52 ; 5 ; start ; len w control-id ESC\
```

Where:

start The location (starting from 1) of the first character to be selected.

len The number of characters that are to be selected.

Using the Clipboard

You can use the clipboard facilities as described:

To cut the selection in the edit box to the clipboard, use the following AiF escape sequence:

```
ESC_52 ; 6 w control-id ESC\
```

To copy the selection in the edit box to the clipboard, use the following AiF escape sequence:

```
ESC_52 ; 7 w control-id ESC\
```

To paste the clipboard contents into the edit box at the current insertion point, use the following AiF escape sequence:

```
ESC_52 ; 8 w control-id ESC\
```

This is ignored if the clipboard does not contain text.

To clear the current selection in the edit box (i.e., deletes it without placing it in the clipboard.), use the following AiF escape sequence:

```
ESC_52 ; 9 w control-id ESC\
```

Initialising a Multi-line Edit Box

To initialise a multi-line edit box with the contents of the named string list, use the following AiF escape sequence:

```
ESC_52 ; 10 w control-id; string ESC\
```

Where:

string The string list id.

Validated Edit Boxes

Validated edit boxes are single-line edit boxes that may only contain information of a specific type, and are **validated** so that they only contain information of that specific type:

- An integer.
- A date.
- An amount of money.

Once you have defined the type of information that the edit box contains, the contents of the edit box must always conform to the format you have specified. The contents can only be changed to valid formats.

For example, if you create a validated edit box for an integer, then that box only accepts valid integers as input. All other inputs will be ignored (sounding a beep).

Creating Validated Edit Boxes

To create a validated edit box, you need to:

1. Create a normal edit box, defining its size to be consistent with the data it contain. For example, if you want a validated edit box to contain an integer between 100 and 999, you should create the edit box to be 1 character high and 3 characters wide.
2. Attach a **validation** to it, defining the allowed contents of that edit box. You can attach integer validations, date validations or currency validations, depending on the type of data required. These validations are described in the following sections.

Validated edit boxes only allow single inputs, on single lines: one date, one number, or one sum of money. If you create a multi-line edit box, then attach a validation to it, the edit box will only allow inputs on the top line. If the initial contents of the edit box do not conform to this format, they are removed. If you destroy an edit box, the associated validation is also destroyed. You can change the validated edit box by attaching a new validation.

Integer Validations

To attach an integer validation to an edit box (defining that box to contain only integers), use the following AiF escape sequence:

```
ESC_70 ; 1 {; low ; high} w control-id ESC\
```

Where:

- | | |
|-------------------|--|
| low | Minimum allowed value for integers. |
| high | Maximum allowed value for integers. Must be higher than low. |
| control-id | Control id of edit box. |

Note: if you specify a **low** parameter, you must also specify a **high** parameter.

If **low** and **high** are both zero, or are not specified, then there are no limits to the integer.

Example

To create an edit box called **emp-nos**, use the following AiF escape sequence:

```
ESC_50 ; 10 ; 10 ; 1 ; 2 w emp-nos ; 17 ESC\
```

To then attach a validation to that box, such that the box only contains valid integers between the values of 1 and 32, use the following AiF escape sequence:

```
ESC_70 ; 1 ; 1 ; 32 w emp-nos ESC\
```

Date Validations

To attach a date validation to an edit box, use the following AiF escape sequence:

```
ESC_70 ; 2 { format } w control-id ESC\
```

Where:

- format** The format of the date information:
- 1 = long date format (for example, Monday, 20 June, 1996).
 - 2 = short date format (for example, 20/06/95).
 - 3* = Abbreviated date (for example, 20/06 - defaults to current year).

control-id Control id of edit box.

The format for dates is defined by Windows. To change this, run the **International** program within Windows Control Panel.

The host always stores dates in the following format:

```
dd/mm/yyyy
```

irrespective of the user's national settings. This increases simplicity over national borders.

Special Date Strings

When displaying date information, you can pass a number of special strings that relate to the current date:

```
yesterday
today
tomorrow
next <monday/tuesday/wednesday/thursday/friday/saturday/sunday>
last <monday/tuesday/wednesday/thursday/friday/saturday/sunday>
```

These strings are not case-sensitive.

Changing the date

You can use the following key presses to change the date within an edit box:

Up Arrow Add a day to the date.

Down Arrow	Subtract a day from the date.
PageUp	Add a month to the date.
PageDown	Subtract a month from the date.
Home	Set the date to today (the current date).

When altering the month, the day of month is adjusted within the bounds of the month. For example, adding a month to **31/01** gives **28/02**.

Note: when within a date-validated edit box, you cannot use the PageUp and PageDown keys to scroll back/forward through the current session's terminal backpages.

Edit Examples

To display an edit box called **payday**, containing the date for the next Friday from the current date, use the following AiF escape sequence:

```
ESC_50 ; 5 ; 5 ; 3 ; 10 w payday ; next friday ESC\
```

To check that the date information in an edit box called **date** is valid, use the following AiF escape sequence:

```
ESC_70 ; 2 w date ESC\
```

Currency Validations

To attach a currency validation to an edit box, use the following AiF escape sequence:

ESC_70 ; 3 w control-id {; format} ESC

Where:

control-id Control id of edit box.

format The currency format – see Defining Currency Format below.

The default currency format is defined by Windows. To alter this, run the **International** program within Windows Control Panel.

Defining Currency Format

You can use the **format** parameter to define a currency format. This format consists of a series of special characters based on the Visual Basic formatting commands, as follows:

Symbol	Meaning
!	Display currency symbol.
#	Display zero or more digits if before a decimal point. Display up to 1 digit if after the decimal point.
0	Display one or more digits; or 0, for leading and trailing zeros.
.	Display a decimal point.
,	Allow the triad separator between triples of digits.
%	Percentage display (number is multiplied by 100, and suffixed with a % sign). This cannot be used with the ! symbol.
-	Force a “-” symbol before negative numbers (the default).
+	Force a “+” symbols before positive numbers.
()	Enclose negative numbers in parentheses.
“string”	Allow literal string. This is collected into one complete string, placed at the end of the currency string. (for example “Gross”, “per annum”).

Currency Examples

To format a currency displayed as a currency symbol, followed by zero or more digits, then a decimal point, then a trailing zero or one digit, use the following string:

!#.0

So to attach a currency validation in this format, use the following AiF escape sequence:

ESC_70 ; 3 w edit ; !#.0 ESC/

The following table shows how particular sums can be represented:

Sum	Format		
	#	!#. #	(#.00) “ pounds”
1.2	1	£1.2	1.20 pounds
12.52	13	£12.5	12.52 pounds
-23.532	-24	-£23.5	(23.53) pounds

Note: Blank edit boxes are always displayed as empty, despite any formatting to the contrary.

Static Labels

You use a static label as a means of getting proportional text on the screen.

To create a static label of a given size, use the following AiF escape sequence:

```
ESC_53 ; y ; x ; h ; wid {; vis} {; en} {; font} {; pos} {; bord} w control-id {; text} {; face} ESC\
```

Where:

y	Y co-ordinate of top of label.
x	X co-ordinate of left of label.
h	Height of label, in rows.
wid	Width of label, in columns.
vis	1 = create hidden. 2* = create visible.
en	1 = initially disabled. 2* = initially enabled.
font	Font: 1* = Terminal. 2 = System. 3 = 10pt Helvetica. 4 = 8pt Helvetica. 5 = Font face name (see face).
pos	1 = left aligned. 2 = right aligned. 3* = centred.
bord	1* = no border. 2 = border.
control-id	Control id.
text	Text to be displayed.
face	Font face name (only if font = 5) e.g. "Times New Roman".

Changing the Windows pointer

It is now possible to change the Windows pointer (cursor) style via the AiF, although this will only apply to the terminal window. Any GUI controls will over ride this style while the pointer is over the area.

Use the following AiF sequence:

ESC_56 ; arrow w ESC

Where:

Arrow

- 0 Restore Standard Pointer.
- 1 Arrow.
- 2 Wait.
- 3 Cross.
- 4 I Beam.
- 5 Icon.
- 6 Up arrow.
- 10 Size.
- 11 Size NESW.
- 12 Size NS.
- 13 Size NWSE.
- 14 Size WE.

Creating a Modal Message Box

A modal message box is a dialog box with a multi-line message, a caption, optionally a bitmap to the left of the message, and one of a variety of standard button combinations. To create a modal message box use:

ESC_91 ; style ; y ; x ; rtn w caption ; message ; spec ; help ; context ESC

Where:

- style** Button Style. The buttons set as default (i.e. those that respond when the ENTER key is pressed) are marked with an asterisk.
- 1: OK + cancel(*).
 - 3: OK(*) + cancel.
 - 5: yes + no + cancel(*).
 - 7: yes(*) + no + cancel.

	9: yes + no(*) + cancel.
	11*: OK.
	13: yes(*) + no.
	15: yes + no(*).
y	Y co-ordinate of message box.
x	X co-ordinate of message box.
rtn	*1 -send response on button click. 2 - do not send response.
caption	Caption (title).
message	Message itself. Separate each paragraph with a CR.
spec	Decoration button spec. See page 39 for further information.

Note: Previous options of this function are still supported but should be changed to reflect the revised options as above.

Positioning the Box

If **x** and **y** are present, and greater than zero, the message box is positioned so the top left of the box coincides with the screen pixel co-ordinate of the top left pixel of the identified character cell in the terminal window. If this forces part of the message box off screen, it is moved if possible to get the whole box on screen.

If either **x** or **y** are 0 or omitted, the message box will be centred on the terminal window.

Returning Values to the Host

The result is returned to the host as

`<STX> n <CR>`

Where:

n 1 = 'no'.

2 = 'yes' or 'OK'.

3 = 'cancel', escape pressed, or msg box closed.

Returning Values to the Host Example

To create a Modal message box at 10,10, with a title, 1 line of text, and Yes/No/Cancel/Help buttons, use:

`ESC_91 ; 6 ; 10 ; 10 w MBox Test ; Click any button - Cancel has focus ESC\`

Status Bar

You can use AiF sequences to modify the status bar for your application:

- Hiding/showing the status line.
- Displaying your own text messages.
- Dividing the status line into panes, and setting the contents of each pane.

The following sections describe how to use these functions.

Hiding/Showing the Status Line

To hide/show the status line, use the following AiF escape sequence:

```
ESC_92 ; 3 ; {show} w ESC\
```

Where:

show 1 = hide status line.
 2* = show status line.

Hiding the status line will increase the area available for the terminal window display.

Setting Status Line Text

To set the main text of the status line, use the following AiF escape sequence:

```
ESC_92 ; 1 ; {timeout} w text ESC\
```

Where:

timeout Number of seconds message is to remain on screen.
 (*=no timeout - message remains until the next message is sent).

text The text of the status line.

For example, to send the text “Press F1 for help” to the status line, use:

```
ESC_92 ; 1 w Press F1 for help ESC\
```

Setting Status Line Pane Contents

To set the contents of one or more panes, use the following AiF escape sequence:

```
ESC_92 ; 2 ; pane; contents ; w ESC\
```

Where:

pane Pane number (1, 2 or 3).

contents Pane contents:
1 = empty pane.
2 = num. Lock status.
3 = caps lock status.
4 = time (hh:mm format).
5 = date (dd-mmmm-yy format).
6 = cursor position (row:column format).

To set multiple panes in one escape, repeat pairs of integer arguments.

Setting Status Line Pane Example

To set 3 panes as follows:

1 = cursor position.
2 = num lock status.
3 = empty.

Use the following AiF escape sequence:

```
ESC_92 ; 2 ; 1 ; 6 ; 2 ; 2 ; 3 ; 1 w ESC\
```

Commands for menus, toolboxes and toolbars

Commands are controls associated with menus, toolboxes or toolbars. Commands can be associated with text (for use in menus), or with button images (for use in toolbars and toolboxes). You must create commands to put in toolbars, boxes and menus before you create the toolbars, boxes and menus.

Once a command has been created, you make it visible to the user by adding it to a 'command container' object, such as a toolbar, and then make the toolbar visible.

Note: Commands in use in menus should not be used to load application menus as these will not work.

Creating Commands

To create or add a command definition, use the following AiF escape sequence:

```
ESC_33 ; 5 ; type {; help} w c-id {; menu} {; spec} {; stat} {; wfile} {; wcont} {; r-id} ESC\
```

Where:

type	Command type (only relevant for toolboxes and toolbars) and initial state: 1 = pushbutton-type command, disabled. 2 = pushbutton-type command, enabled. 3 = check box-type command, unchecked, disabled. 4 = check box-type command, unchecked, enabled. 5 = check box-type command, checked, disabled. 6 = check box-type command, checked, enabled. 7 = check box-type command, indeterminate, disabled. 8 = check box-type command, indeterminate, enabled.
help	WinHelp command. See page 90 for details of invoking Windows help.
c-id	Id for new command.
menu	Menu text. i.e., the text to be used when this command is added to a menu.
spec	Image specification when this command is used in a floating toolbox or toolbar. The same image is used for toolboxes and toolbars. The images for all button states (enabled, checked etc.) is automatically computed. See Appendix A, Describing Images for details on specifying an image.
stat	Status line prompt. This text will be displayed in the status line when the user selects the command (e.g., by holding down a button in a toolbox).
wfile	WinHelp filename - used when user requests help for this command.

wcont WinHelp context.
r-id Radio command group id.

Changing Command Type and State

To change a command type and state, use the following AiF escape sequence:

ESC_33 ; 6 ; type w c-id ESC

Where:

type New command type and state. Values are 1 - 8, as for creating commands, see page 80 for details.
c-id Command id.

All toolboxes and toolbars that currently show the command are redrawn to reflect the new state.

Reading a Command

To read the current type and state for a given command, use the following AiF escape sequence:

ESC_33 ; 7 w c-id ESC

Where:

c-id The command id.

The format of the value returned is as follows:

<STX> state<CR>

Where:

state The current state (1..8).

Setting Command Images in Toolbars/Toolboxes

To explicitly set the images to be used to show commands in toolboxes and toolbars, use the following AiF escape sequence:

```
ESC_33 ; 8 w c-id ; norm ; dis ; check ; indet ; norm2 ; dis2 ; check2 ; indet2 ESC\
```

Where:

c-id	Command id.
norm	Image spec for toolbar, 'normal'.
dis	Image spec for toolbar, 'disabled'.
check	Image spec for toolbar, 'checked'.
indet	Image spec for toolbar, 'indeterminate'.
norm2	Image spec for toolbox, 'normal'.
dis2	Image spec for toolbox, 'disabled'.
check2	Image spec for toolbox, 'checked'.
indet2	Image spec for toolbox, 'indeterminate'.

This allows different images to be used for toolbars and toolboxes, and/or different images for the different command states.

This should be done before adding the command to a toolbox/toolbar. It should not be used to dynamically change the button once visible.

Adding a New Command Group

To add a new command group, use the following AiF escape sequence:

```
ESC_33 ; 9 w group-id ESC\
```

Where:

group-id	The command group id.
-----------------	-----------------------

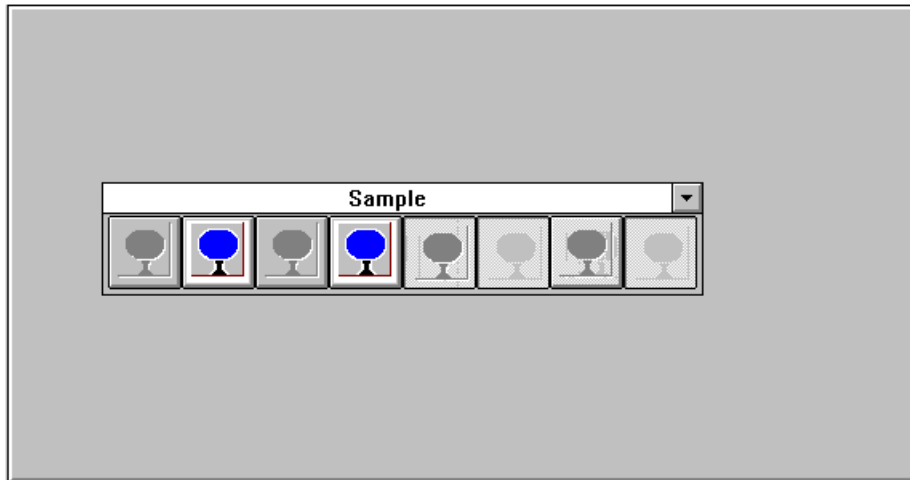
New Command Group Example

The following shows a toolbox containing eight buttons (commands), in all eight possible states.

All the commands use the same bitmap image. The toolbox does not have focus, and has a minimise box attached. The code is implemented using the following sequences

```
ESC_33 ; 1 ; 10 ; 10 ;; 2 w tbox ; Sample ; c:\windresstiger.ico ESC\  
ESC_33 ; 5 ; 1 w tcom1 ;; file=c:\bitmaps\tree.bmp ESC\  
ESC_33 ; 5 ; 2 w tcom2 ;; file=c:\bitmaps\tree.bmp ESC\  
(etc ...)  
ESC_33 ; 4 w tbox ; tcom1 ESC\  
ESC_33 ; 4 w tbox ; tcom2 ESC\  
(etc ...)  
ESC_33 ; 2 w tbox ESC\  

```



Toolbars and Toolboxes

You can use AiF escape sequences to create and use floating toolbars and toolboxes. The following sections describe how to use these functions:

- Creating a floating toolbox.
- Hiding/showing a toolbox.
- Creating a toolbar.
- Adding a button to a toolbar/toolbox.

Before creating a toolbox or toolbar, you must create **commands** to put in those toolboxes, see page 80 for details of commands.

Creating a Floating Toolbox

To create a floating toolbox, use the following AiF escape sequence:

```
ESC_33 ; 1 ; x ; y {; mod} {; border} {; min} {; orig} w t-id {; title} {; icon} ESC\
```

Where:

- x** x co-ordinate of top of toolbox (see orig, below).
- y** y co-ordinate of left of toolbox (see orig, below).
- mod** No longer supported – defaults to modeless.
- border** No longer supported – defaults to no border.
- min** No longer supported – defaults to do not display a minimise box.
- orig** Window origin:
 - 1 = relative to the screen, in pixels.
 - 2 = relative to the application window, in pixels.
 - 3* = relative to the main terminal window, in character cells.
- t-id** Unique toolbox id.
- title** Title text for toolbox.
- icon** Icon file name, to be used when toolbox is minimised. Must be a .ico file.
 - If not given, a default icon will be used. Only relevant if a minimise box is displayed.

Note: the toolbox will be hidden by default - you have to show it to display the toolbox on the screen.

Hiding/Showing a Toolbox

To hide or show a toolbox, use the following AiF escape sequence:

```
ESC_33 ; 2 ; {show} w control-id ESC\
```

Where:

- show** 1 = hidden.
- 2 = minimised.

3* = normal.

4 = maximised.

control-id Toolbox control id.

Creating a Toolbar

To create a toolbar, use the following AiF escape sequence:

```
ESC_33 ; 3 ; {bar} w control-id ESC\
```

Where:

bar 1* = toolbar initially empty.

 2 = base toolbar on default HostAccess toolbar.

control-id Toolbar control id.

Adding a Button to a Toolbar/Toolbox

You must create the toolbar/toolbox before you can add a button. The button command must already exist before you create the button. You should add all commands to a toolbox/bar whilst it is hidden, and then show it at the end. To add a button to a toolbox/toolbar, use the following AiF escape sequence:

```
ESC_33 ; 4 { place} ; gap w t-id ; c-id ESC\
```

Where:

place Where to place new button: 1* = to right of last; 2 = start new row of buttons.

gap Gap between new button and previous button. If adding to same row, this is the number of pixels of gap inserted to the left of the new button. When starting a new row, this is the number of pixels of gap to insert above the new button.

t-id Toolbox/toolbar id.

c-id Command id (the id of an existing command). May be an internal command.

Toolbar Button Example

The following example:

- Creates a floating toolbox named **tbox**, with header text “Sample”.
- Creates two commands, **tcom1** and **tcom2**, and links these commands with bitmap images.
- Adds both **tcom1** and **tcom2** to the floating toolbox **tbox**.
- Shows the floating toolbox.

This can be coded as follows:

```
ESC_33 ; 1 ; 10 ; 10 w tbox ; Sample ESC\
```

```
ESC_33 ; 5 ; 2 w tcom1 ;; file=c:\bitmaps\tree.bmp ESC\
```

```
ESC_33 ; 5 ; 2 w tcom2 ;; file=c:\bitmaps\question.bmpESC\
```

```
ESC_33 ; 4 w tbox ; tcom1 ESC\
```

ESC_33 ; 4 w tbox ; tcom2 ESC\
ESC_33 ; 2 w tbox ESC\

Menus

You can use AiF escape sequences to create controls for Windows-style menus on the menu bar for your application. To use menus, you must first create a set of **commands** to place in the menus you create, see page 80 for details of commands.

Creating Menus

To create a new menu, use the following AiF escape sequence:

```
ESC_33 ; 10 ; {dis} w m-id ; title ; c-id1 ... ESC\
```

Where:

- dis** 1 = disabled.
 2* = enabled.
- m-id** New menu id.
- title** Menu title.
- c-id1 ...** Command ids, or menu ids (see below) to be added to the menu. To add a separator, skip a command id (i.e., 2 semicolons with nothing between).

You can use this AiF escape sequence to create hierarchical menus (that is, menus containing menus), by including the name of a pre-defined menu as one of the command IDs.

Menu Example

To create a menu called **fonts**, containing the commands **bold** and **italic**, and the sub-menu **size**, use the following AiF escape sequences:

```
ESC_33 ; 10 w size ; Font Sizes ; eight ; ten ; twelve ESC\
ESC_33 ; 10 w fonts ; Character Fonts ; bold ; italic ; size ESC\
```

The sub-menu **size** contains the commands **eight**, **ten** and **twelve**.

Displaying Menus

To place pre-defined menus in the menu bar, use the following AiF escape sequence:

```
ESC_33 ; 11 ; c-num w c-id1 ... menu-ids ESC\
```

Where:

- c-num** Number of command IDs to be inserted on help menu.
- c-id1 ...** Command IDs to be added to help menu.
- menu-ids** IDs of menus to be installed in menu bar.

The host menus will be inserted to the left of the Help menu, but left aligned (that is to the right of all the inbuilt non-help menus).

You can also give command IDs to be added to the help menu, in a separate section.

Removing Menus

To remove menus from the menu bar, pass the control IDs of the menus to be inserted in the menu bar, in the order you want them. All host menus that are not named will be removed from the menu bar if already there.

So to remove all host menus, use this AiF escape sequence, without naming any menus.

Enabling/Disabling Menus

To enable/disable a whole menu, use the following AiF escape sequence:

```
ESC_33 ; 12 ; dis w menu-id ESC\
```

Where:

dis 1 = disabled, 2* = enabled.

menu-id Menu id.

Changing Fonts

To change the font of characters displayed on your terminal, use the following AiF escape sequence:

ESC_93 {; size} w {name} ESC

Where:

size New font size. * = current font size.

name Name of font. * = current font.

For example, to switch display font to Letter Gothic 18 point, use the following AiF escape sequence:

ESC_93 ; 18 w Letter Gothic ESC

Windows automatically selects the closest available match to the font you select.

This sequence is equivalent to using the Font... option of the Configure menu. If you change the font size, this automatically switches Maintain Aspect Ratio on. If you have set Snap To Frame or, Best Fit, changing the font size has no effect.

Invoking Windows Help

To invoke Windows Help, use the following AiF escape sequence:

ESC_94 ; 1 ; invoke w file ; context ESC

Where:

invoke Specifies how to invoke Windows Help:

1*= HELP_CONTEXT

2 = HELP_CONTEXTPOPUP

3 = HELP_CONTENTS

4 = HELP_KEY

5 = HELP_PARTIALKEY

6 = HELP_COMMAND

7 = HELP_HELPONHELP

8 = HELP_QUIT

file Help file name. If missing, the HostAccess help file is used.

context Either a help context number (if invoke = 1 or 2), a help keyword or partial keyword (if invoke =4 or 5), a help macro string (if invoke =6), or is ignored.

See the Microsoft Windows Help Authoring Guide documentation for details of **Winhelp()** and the **HELP_...** functions.

Timed Events

To return a notification after a set time period has elapsed, use the following AiF escape sequence:

ESC_95 ; delay ; t-event ; a-event w ESC

Where:

delay	The number of seconds to wait before triggering.
t-event	1 = turn timed events off. 2 = turn on one timed event. 3 = turn on regular timed events (every time interval).
a-event	1 = turn activation events off. 2 = turn activation events on.

The return notification takes the following format:

<STX > Type <CR> 1 <CR>

Where

Type	A string: “TI” to signal a timed event. “AC” to signal that HostAccess has been made active (gained focus).
-------------	---

ActiveX (COM) Integration

An ActiveX object is a component program object that can be re-used by many application programs within a computer or among computers in a network. The technology for creating ActiveX objects is part of Microsoft's overall ActiveX set of technologies, chief of which is the Component Object Model (COM). ActiveX objects can be downloaded as small programs or applets from Web pages, but they can also be used for any commonly needed task by an application program in the latest Windows and Macintosh environments. In general, ActiveX objects replace the earlier OCX's (Object linking and embedding custom objects). An ActiveX object is roughly equivalent in concept and implementation to the Java applet.

An ActiveX object can be created in any programming language that recognizes Microsoft's Component Object Model (COM). An ActiveX object is a component or self-contained program package that can be created and reused by many applications in the same computer or in a distributed network. The distributed support for COM is called the Distributed Component Object Model (DCOM). In implementation, an ActiveX object is a Dynamic Link Library (DLL) module. An ActiveX object runs in what is known as a container, an application program that uses the Component Object Model program interfaces. This reusable component approach to application development reduces development time and improves program capability and quality.

HostAccess will access the ActiveX objects in the same way as it has previously done for its standard internal controls. The host programmer prints a unique escape code that is interpreted by HostAccess and an action is performed. These objects can be positioned on the HostAccess emulation screen at a given X & Y co-ordinate and scaled to a size of a number of characters by number of rows. The escape sequences used to load and manipulate these objects follow the same standards as previous escape cod

Creating ActiveX (COM) Objects/Controls

There are 2 types of objects that can be 'created', objects that can be positioned on the HostAccess screen and those that cannot. The first tend to be GUI controls written for Visual Basic like edit controls, radio buttons, list boxes, browser objects and the later are automation objects that provide a programming interface to control applications like MS Word and MS Excel.

To create an object, use the following AiF escape sequence:

ESC_201 ; object ; y ; x ; h ; wid ; status w control-id ; prog-id ; caption ESC

Where:

object	Type of Object: 0 = Automation Object 1= GUI Object
y	y co-ordinate of top of the Object.
x	x co-ordinate of left of the Object.
h	Height of the Object, in rows.
wid	Width of the Object, in columns.
status	Status of Object creation: 1 = Return a status response 2* = Do not return a status response
control-id	Control ID - must be unique, and may not be "root".
prog-id	Fully qualified class name e.g. Pixel.Button
caption	Text value that the object may use. Typically used with label and edit controls.

The following response will be sent to the host application, if the status parameter was set to 2:

<STX> value <CR>

Where:

<STX>	Is the start of text character (ASCII decimal value 002).
value	Is the actual value returned. If an object is created, the value returned is 1. If an object fails to be created, the return value will be 0.
<CR>	Is a carriage return (ASCII decimal value 013).

Registering and Using ActiveX (COM) Objects/Controls

Make sure all ActiveX controls (*.ocx) exist and are registered on the client machine.

To begin registration of an ActiveX control, open a command prompt.

NOTE: If you are on a Windows 7 or higher operating system AND UAC is enabled, start the command prompt using "Run as Administrator" to use regsvr32.exe correctly.

To register the ActiveX control on a 32-bit (x86) system,

- move <name of control>.ocx to C:\Windows\system32
OR
- move <name of control>.ocx to C:\WINNT\system32
- then call <exe path>\regsvr32.exe <ocx path>\<name of control>.ocx from the command prompt

To register the ActiveX control on a 64-bit (x64) system,

- move <name of control>.ocx to C:\Windows\sysWOW64
OR
- move <name of control>.ocx to C:\WINNT\sysWOW64
- then call <exe path>\regsvr32.exe <ocx path>\<name of control>.ocx from the command prompt.

NOTE: <exe or ocx paths> are only required if either differ from C:\Windows (or WINNT)\system32 or C:\Windows (or WINNT)\sysWOW64, otherwise the <exe path>\ or <ocx path>\ may be omitted. Also replace <name of control> with the actual name of the OCX control in the instructions above.

Chart control Example:

To create a Chart Control on the HostAccess screen with the unique name of CHART at a position of Row 5 Column 10 with a depth of 12 rows and width of 40 characters use the following AiF sequence:

```
ESC _ 201 ; 1 ; 5 ; 10 ; 12 ; 40 w CHART ; MSChart20Lib.MSChart.2 ESC \
```

Grid control Example:

To create a Grid Control on the HostAccess screen with the unique name of GRID at a position of Row 5 Column 10 with a depth of 12 rows and width of 40 characters use the following AiF sequence:

```
ESC _ 201 ; 1 ; 5 ; 10 ; 12 ; 40 w GRID ; MSFlexGridLib.MSFlexGrid.1 ESC \
```

NOTE: There are known licensing issues with the above ActiveX controls. If either control does NOT appear in their respective demos or macro samples when running HOSTACCESS.DEMO from the TCL or from your PICK code, please refer to Microsoft's documentation for possible licensing issues for these controls at <http://support.microsoft.com/default.aspx?scid=kb:en-us:318597>.

Visit our Host Access forums at <http://forums.roguewave.com> to view and share what other users' are doing with macro and PICK samples.

Miscellaneous Example:

To create a word document object outside the HostAccess Screen use the following AiF sequence:

```
ESC _ 201 ; 0 w WORD ; Word.Document.8 ESC \
```

Executing Methods

To call a method of an object you will need to know what parameters to supply (if any). Any number of parameters will be accepted by HostAccess, as it does not validate your syntax at execution time. You can also pass additional objects into methods using the relevant Control ID. E.g. If a method required an image object as a parameter you can send it the Control ID of the image object, which has previously been created. HostAccess will convert this object into the correct “type” before executing the method.

To execute a method, use the following AiF escape sequence:

ESC_204 w control-id ; method ; param1 ; param2 ; param(n) ESC

Where:

control-id	Is the Control ID of the object.
method	Is a function that is associated with the object. This can sometimes be called a “member function”
param1-n	These are optional parameters that are associated with the method.

The following response will be sent to the host application:

<STX> value <CR>

Where:

<STX>	Is the start of text character (ASCII decimal value 002).
value	Is the actual value returned. If an object is returned by the method then HostAccess will convert this to a Control ID. You can then use the control as though it was an automation object. The Control ID assigned by HostAccess is based on the control ID that returns the object. If the control ID is “CHART” and you call a method which returns another object it will be called “CHART_0” or “CHART_1” if CHART_0 has already been used.
<CR>	Is a carriage return (ASCII decimal value 013).

Example:

To increase the month of the CALENDAR object using the “NextMonth” method use the following escape sequence:

*ESC_204 w CALENDAR ; NextMonth ; -1 ESC *

Getting a Property Value

To get a property value, use the following AiF escape sequence:

*ESC_202 w control-id ; property ; param1 ; param2 ; param(n) ESC *

Where:

control-id	Is the Control ID of the object.
Property	This is the property associated with the object.
param1-n	Any additional parameters that are used to utilize the property.

The Get Property Value response will be:

<STX> value <CR>

Where:

<STX>	Is the start of text character (ASCII decimal value 002).
value	Is the actual value returned. If an object is returned by the method then HostAccess will convert this to a Control ID. You can then use the control as though it was an automation object. The control ID assigned by HostAccess is based on the control ID that returns the object. If the control ID is "CHART" and you call a method which returns another object it will be called "CHART_0" or "CHART_1" if CHART_0 has already been used.
<CR>	Is a carriage return (ASCII decimal value 013).

Example:

To return the BackDrop Fill Style of the CHART object use the following AiF sequence:

*ESC_202 w CHART ; BackDrop ESC *

Setting a Property Value

To set a property value, use the following AiF escape sequence:

*ESC_203 w control-id ; property ; param1 ; param2 ; param(n) ; value ESC *

Where:

control-id	Is the Control ID of the object.
Property	This is the property associated with the object
param1-n	Any additional parameters that are used to set the property.
value	Value to set.

There may be multiple parameters so the last value in the escape code will be taken as the value.

Example:

To set the “ChartType” property of the CHART object to Type 6 use the following AiF sequence.

*ESC_203 w CHART ; ChartType ; 6 ESC *

Enabling Events for an Object/Control

One of the problems with using external objects is that they have been written for use with PC applications where the Event handling is controlled locally on the PC. What happens is that objects are written to support lots of different event types like “mouseover”, “Got Focus”, “Lost Focus”, “mouseup”. In a Visual Basic application these events are just discarded with no noticeable performance loss. When you then move this concept to the world of HostAccess you will see the problem when the events are transmitted back to the Host. Imagine moving the mouse over a button and the control sending a few hundred events over the network or down a serial cable to the host. What about the poor host, the volume alone would probably crash the input buffer. All that said we have resolved these issues by implementing an additional stage into the event handling to disregard and stop transmitting all the unwanted events.

To enable/disable events of an object, use the following AiF escape sequence:

*ESC_203 w control-id ; type ; event ESC *

Where:

control-id	Is the Control ID of the object.
type	“ENABLEEVENT” or “DISABLEEVENT”.
event	The event to be enabled/disabled.

Once the object event has been enabled you must enable HostAccess ActiveX event reporting.

Example:

To enable a Click event for the BUT1 object use the following AiF sequences:

*ESC_203 w BUT1 ; ENABLEEVENT ; Click ESC *
*ESC_15 ; 3 ; 16 w BUT1 ESC *

ActiveX Event Reporting

To enable/disable HostAccess ActiveX Event reporting, use the following AiF escape sequence:

*ESC_15 {; enable} ; 16 w control-id ESC *

Where:

enable	1 = Disable events (discards outstanding stacked events). 2*= Enable events. 3 = Stack events (recommended for ActiveX use).
control-id	Is the Control ID of the object or the Control Group ID.

When an ActiveX event is reported to the host, information about that event is sent in the following format:

<STX> WC <CR> id, 16, event, noofparams, params ... <CR>

Where:

<STX>	Is the start of text character (ASCII decimal value 002).
WC	Literal characters.
<CR>	Is a carriage return (ASCII decimal value 013).
id	Control ID of the control associated with the event.
16	Literal characters.
event	The event name i.e. Click.
noofparams	The number of parameters that are supplied with the event.
params	The parameters that follow.

Destroying an ActiveX Object/Control

To destroy a named control, string list or control group, use the following AiF escape sequence:

```
ESC_10 {; delete} w control-id ESC\
```

Where:

delete	Use only if ID is that of a control group: 1 = do not delete controls inside group. 2* = delete all controls in group.
control-id	Control ID, string list ID or control group ID.

Destroying a control will flush it from HostAccess memory. The control is immediately removed from the screen. If the specified control currently has focus, or would have focus if the application were the active top level Window, then focus is shifted to the root.

Deleting a control group will by default delete all the controls in that group. To retain the controls, set the delete parameter to 1.

Common Problems

Some common problems you may encounter using the Windows AiF features are described below with suggested solutions.

Sculpting

Sculpted Boxes/Lines do not appear

Check that you have turned the sculpture mode on.

Sculpted Boxes/Lines still appear after application ends

Check you have turned the sculpture mode off.

Control Management

Cannot Change Control Colour

Check that you have the correct control name.

Check that you change the colour **after** displaying the control.

No Response From Clicking a Control

Check that the event has been set up correctly with the correct name.

Check that the control you are using has a response - some do not have an attached event.

Check that the mouse is working correctly.

Check that the correct event number has been enabled for the control.

Secondary Windows

Cannot switch from a secondary window

Check that the window is modeless.

Buttons

No image appears

Check that the image file exists, and is of the correct type. See Appendix A, *Describing Images* for details.

Toolbars and Toolboxes

Toolbars or Toolboxes do not appear

Check that you have specifically set the toolbox or toolbar to show.

Fonts

Control's font is different to the specified font.

When you create a control containing text (for example, an edit box), and you specify the font (for example, Helvetica 8), this font may be changed in your display. This is because Windows substitutes certain font, as specified in the [fontsubstitutes] section of the user's WIN.INI file. For example, Arial may be substituted for Helvetica.

To alter this, alter the font substitution - see your Windows programming guide for details.

Note: If a font is not defined on the user's PC, Windows always tries to match to the closest available font.

Controls and Macros

General ActiveX controls such as MSChart, MSFlexGrid or Calendar (MSCAL) do not ship with Host Access. As a result some macros, i.e. AXFIG1.mcr through AXFIG4.mcr, or demos may not perform correctly without these controls. Install and register these controls on the client machine prior to running the macros or demos.

Please refer to Chapter 2's section titled, **Registering and Using ActiveX (COM) Objects/Controls**, for a detailed solution.

AiF Utilities

The following sections describe how you can use the **Applications interface Facility** (AiF) to exploit the power of library routines and screen manipulation features. HostAccess provides a sophisticated, application driven workstation. While continuing to support existing applications unchanged through industry standard terminal emulations, you can also introduce a PC style user interface to host applications with colour, windows, pop-down menus and many more features.

How AiF Sequences Work

The AiF supports standard ANSI X3.64 compliant ESCape sequences that may be used by host applications to drive the AiF features.

Any host process that can send output to a terminal can also make use of AiF by sending special AiF sequences to HostAccess running on a PC. HostAccess intercepts these sequences and takes the appropriate action (for example, saving a screen image).

Software developers normally define these AiF sequences so that they can be referenced globally as variables by their applications code (either at run-time or compile time).

Types of Sequence

HostAccess expects the AiF sequences to conform to a certain format. Every screen AiF sequence starts with the ESCape character (ASCII decimal value 27). The next two characters in the sequence depend on the type of feature required.

The semi-colon character is used as a delimiter to separate parameters in a sequence.

Note: a common programming error when using AiF sequences is to forget/misplace the delimiters.

Screen Manipulation Sequences

This type of sequence is called an ANSI CSI (Control Sequence Introducer). Generally, these format sequences usually denote that the sequence is being used by HostAccess for colour, box or line drawing, saving/restoring screen images, etc.

This type of sequence requires a terminating character, telling the AiF what facility is required - these are described for each sequence.

Format: left square bracket and equals sign (**[=**), terminated by a lower-case character.

So an entire sequence in this format can be described as:

ESC [= A1 ; A2 ; ... An f

Where:

- ESC** Is an escape character (decimal value 27).
- A1 ... An** Are parameters (typically colour attribute setting, column/row co-ordinate).
- f** Is a lower case terminating character, such as 'x' for box drawing.

Library Sequences

This type of sequence is called an ANSI APC (Application Program Command). Generally, this format denotes that the sequence is being used by the AiF to interface into main DOS library routines, such as Pop-Down Menus.

Format: Underscore '**_**', followed by a capital letter terminated by ESC ****. So an entire sequence in this format can be described as:

**ESC _ B1 ; B2 ; ... Bn F data string ESC **

Where:

- ESC** Is an escape character (Decimal value 27).
- B1 ... Bn** Are parameters. These parameters depends on the AiF sequence.
- F** Is an upper-case letter (such as 'X').
- data string** Is optional data for the AiF, such as box heading text.
- ESC** Is the escape character (decimal value 27), followed by a backslash character '\'

Note: spaces are shown between characters only for the purposes of clarity - these spaces should not be included within the sequence itself.

Sequences Summary

The following screen AiF sequences are supported:

Tailoring the environment	See from page 110.
Set colour.	ESC [= A1 ; A2 ; A3 ;...An m
Switch ANSI colour mode ON.	ESC [= 7 h
Switch ANSI colour mode OFF.	ESC [= 7 l
Detect colour/mono monitor.	ESC [= 6 n
Detect blinking status.	ESC [= 7 n
Open window.	ESC [= Y1 ; X1 ; Y2 ; X2 ; WT ; A1 ; ... ; An u
Close window	ESC [= F v
Window heading.	ESC _ X1 ; A1 ; ... ; An W ...title... ESC \
Window footing.	ESC _ X1 ; A1 ; ... ; An U ...title... ESC \
Load exit keys.	ESC _ Z exit_keys ESC \
Reset pop-down menus.	ESC [= 2 ; SN l
Load pop-down menus.	ESC _ N1 ; SN M H1 ; E1 ; E2 ; .. ; En ESC \
Activate pop-down menu.	ESC [= 22 ; SN ; m ; e ; Y1 h
Close current pop-down menu.	ESC [= 22 l
Reset cascading menus.	ESC [= 2 ; SN l
Load cascading menus.	ESC _ N1 ; SN M H1 ; E1 ; E2 TC CN ; .. ; En ESC \
Activate cascading menu.	ESC [= 22 ; SN ; m ; e ; Y1 h
Close cascading menus.	ESC [= 22 l
Re-set selection boxes.	ESC [= 24 ; SN l
Load selection boxes.	ESC _ SN ; Y1 ; X1 ; DT ; BT ; MT ; MW S H1 ; E1 ; E2 .. En ESC \
Activate selection boxes.	ESC [= 23 ; SN ; En h
Activate selection boxes in previously opened window.	ESC [= 24 ; SN ; En h
Close selection boxes.	ESC [= 23 l
Activate Line input.	ESC [= 25 ; FL ; VL ; SM h

Tailoring the environment	See from page 110.
Activate Box input.	ESC _ Y1 ; X1 ; FL ; BS ; BT ; VL ; SM J text ; title ESC \
Invoke window editor.	ESC [= 26 h
Load exit keys.	ESC _ Z exit-keys ESC \
Push environment.	ESC [= 99 p
Pop environment.	ESC [= 99 q
Display Optimisation	See from page 148.
Save SLOT number N.	ESC [= N p
Restore SLOT N to screen.	ESC [= N q
Push screen image onto SLOT STACK.	ESC [= p
Pop screen image from SLOT Stack.	ESC [= q
Write to FORM number Fn, form version number Fv	ESC [= Fn ; Fv ; 1 s TEXT ESC [= s
Display from FORM number Fn	ESC [= Fn r
Change FORM file name.	ESC _ F DOS_form_file_name ESC \
Clear currently active FORM file.	ESC [= s
Request FORM file version number	ESC [= Fn ; 1 r
Freeze ON.	ESC [= 1 h
Freeze OFF.	ESC [= 1 l
Turn host echo on.	ESC [= 13 h
Turn host echo off.	ESC [= 13 l
Applications enhancement	See page 157.
Draw box.	ESC [= Y1 ; X1 ; Y2 ; X2 ; BT ; A1 ; ... ; An x
Draw Line.	ESC [= Y1 ; X1 ; Y2 ; X2 ; LT ; A1 ; ... ; An z
Display message.	ESC [= C1 ; A1 ; ... ; An w message CR

Applications enhancement	See page 157.
Clear message line.	ESC [= w CR
Force system message line display.	ESC [= 11 h
Force HostAccess status line display.	ESC [= 11 l
Set mode specified by n.	ESC [= 3 ; n h
Reset to user configured screen mode.	ESC [= 3 l
Selects block cursor.	ESC [= 4 h
Selects line cursor.	ESC [= 4 l
Cursor ON.	ESC [= 10 h
Cursor OFF.	ESC [= 10 l
Set screen fill character to character with ASCII value nnn.	ESC [= 12 ; nnn h
Reset fill character to space.	ESC [= 12 l
Switch to PC font table specified by n.	ESC [= 9 ; n h
Reset default font table.	ESC [= 9 l
Suppress screen output outside current window.	ESC [= 5 h
Disable output suppression.	ESC [= 5 l
Centre text in window.	ESC _ Y1 C text ESC \
Macros.	ESC_ sscripttext ESC \
Keyboard control features	See from page 170.
Program Function key n.	ESC _ n K Key data ESC \
Toggle Caps Lock ON.	ESC [= 28 h
Toggle Caps Lock OFF.	ESC [= 28 l
Switch scancode keys ON.	ESC [= 6 ; p h
Switch scancode keys OFF.	ESC [= 6 l
Switch typeahead ON.	ESC [= 20 h

Switch typeahead OFF. ESC [= 20 l

Keyboard control features See from page 170.

Enable command stack. ESC [= 21 h

Disable command stack. ESC [= 21 l

Detect if mouse installed. ESC [= 8 n

Switch mouse monitoring ON. ESC [= 27 ; n h

Switch mouse monitoring OFF. ESC [= 27 l

DOS Integration See from page 182.

Invoke DOS gateway. ESC _ sc ; 0 D Cmd1 ; ... ; Cmdn % keys ; Cmdnn ESC \

Print screen. ESC [= 0 I

Switch OFF direct (slave) printing. ESC [= 4 I

Switch ON direct (slave) printing. ESC [= 5 I

Change current print device. ESC _ L device.name ESC \

Erase single DOS file. ESC _ E filename ESC \

Request working DOS run directory. ESC [= 9 n

Verify DOS path. ESC _ G path ESC \

Windows Integration See from page 194.

Displays an image. IMAGE /I filename {/T title} {/Z zoom} {/F}

Closes an image application. ESC _ x AP ESC \

Control Window state. ESC _ ST c AP ESC \

Start any Windows program on the desktop. ESC _ ST e PN ESC \

Detect if Windows application is running. ESC _ a AP ESC \

Send keys in DOS keyboard stacker format to specified Windows application. ESC _ k AP % keys ESC \

Dynamic Data Exchange	See from page 206.
Close a DDE link already established with Initiate DDE .sequence	ESC _9d SN;TP ESC \
Send commands to server application	ESC _ 2 ; TM d SN ; TP ; MA ESC \
Open a DDE channel with a server	ESC _ 1d SN;TP ESC \
Pass data to server	ESC _ 3; TM d SN;TP;IT;ST ESC \
Retrieve data from server	ESC _ 4; TM d SN;TP;IT ESC \
Miscellaneous Facilities	See from page 200.
Close HostAccess from host.	ESC _ X ESC \
Request serial number.	ESC [= 1 c
Returns information about HostAccess and its run-time environment.	ESC [= 10 n
Request Printer Information	ESC _ 84 w ESC \
Request the PC Date and/or Time	ESC _ 84 w ESC \
Request the Computer Name and/or User Name	ESC _ 84 w ESC \
Request an Environment Variable value	ESC _ 84 w ESC \
Send screen to host system.	ESC [= 2 I, ESC [= 2 ; n I
Change emulation	ESC [= n {
File transfer	ESC_mode;hostdriven1;append;0;protocol;ist direction local ;remote esc\

Tailoring the Environment

A user of an application should easily and intuitively understand how it works and interacts. Presentation (the user interface) is the most important part of an application's acceptability. The AiF enables developers to design sophisticated and friendly application user interfaces without the need for complicated coding.

Interface aspects that can make an immediate impact on users are:

- Colour, see below.
- Windows, see page 114.
- Menus, see page 117.
- Selection boxes, see page 132.
- GUI: See Chapter 2, AiF TOOLKIT for further details.

Using Colours

HostAccess supports ANSI standard colour sequences in most of its Terminal Emulations.

Standard terminal video attributes are mapped into colour by default (such as bold into red on black), enabling existing applications to use colour without modification.

We have also defined a series of ANSI compatible colour sequences so you can use any PC colour from within any application regardless of the terminal type being emulated by HostAccess.

AiF Sequence - Using Colours

ESC [= A1 ; ... ; An m

Where **A1 ... An** can have the following values:

0	All attributes off (colours are reset to light Gray text on Black background).		
1	Intense on		
2	Intense off		
22	Intense off		
7	Reverse on		
27	Reverse off		
30	foreground to Black	40	background to Black
31	foreground to Red	41	background to Red
32	foreground to Green	42	background to Green
33	foreground to Brown	43	background to Brown
34	foreground to Blue	44	background to Blue
35	foreground to Magenta	45	background to Magenta
36	foreground to Cyan	46	background to Cyan
37	foreground to Light Gray	47	background to Light Gray

The 16 PC foreground colours are achieved by using the 8 colours above with or without the intense bit set.

The Intense Bit Set

Colour	Attribute Value	Colour Value	Attribute
Black	30	Dark Gray	30;1
Red	31	Light Red	31;1
Green	32	Light Green	32;1
Brown	33	Yellow	33;1
Blue	34	Light Blue	34;1
Magenta	35	Light Magenta	35;1
Cyan	36	Light Cyan	36;1
Light Gray	37	White	37;1

To set the screen colours back to the current HostAccess default colours use the AiF sequence:

ESC [0 m

This restores the colours to the state they were in when HostAccess was loaded or to the colours set by the last ESC [=90..m sequence (described immediately below).

Special values may be assigned to the *first* attribute in the colour sequence to change the normal text colour and the default colour settings for application driven AiF Menus.

The following attribute values apply:

- A1 = 90** Changes the default normal colour, used for clear screens, clear to end of line, etc.
- A1 = 91** Changes the Window and Box Shadow colours.
- A1 = 97** Changes the main Menu colour.
- A1 = 98** Changes the selection character colour in Menus.
- A1 = 99** Changes the Menu highlight 'bar' colour.

(Remember, if the first attribute is not one of the above values, the current colour attributes will be changed.)

Resetting the colour parameter to the default setting

To reset the appropriate parameter to the default setting, use one of the following sequences:

```
ESC [ = 90 m
ESC [ = 91 m
ESC [ = 97 m
ESC [ = 98 m
ESC [ = 99 m
```

Using Colours Example - DOS AiF

To reset the user's default foreground and background screen colours to white text on a Blue background use the following command sequence:

```
ESC [ = 90;0;1;37;44 m
```

and then clear the screen.

Note: attribute parameters change only one component each, e.g. 'ESC [= 37 m' changes the foreground colour to white but does not affect the background colour, intensity.

If colour is simply used to highlight (say) an error message, and the you do not want to reset the current colour attributes, you can simply open a window to display the message. Within this window the colours may be configured without affecting the current screen's attributes. Once the user has read the message display in the window, the application program simply closes the window.

Switching ANSI Colour Mode On/Off

Use this AiF sequence to tell HostAccess to support ANSI standard colour sequences in all of its Terminal Emulations.

Normally attributes such as flashing, intense and reverse are mapped through HostAccess's internal tables into special colours so as to give monochrome applications some immediate colour (without the need to change these applications). In ANSI Colour Mode, video attributes are applied literally.

Use the following sequence:

ESC [= 7 h Switches ANSI Colour Mode on.

ESC [= 7 l Switches ANSI Colour Mode off.

Using Windows

AiF's programmable windows is one of HostAccess's most powerful features. Properly used it can liberate applications from the restrictions of only being able to display information on one screen at a time.

AiF windows allow applications to open up a 'virtual' screen of any size anywhere on the current screen. All output sent by the application to the PC's screen will be displayed within this window. Cursor addressing is now relative to the top left-hand corner of this window. If the application clears a screen, changes a video attribute, or the fore/background colours, etc., these will only affect the area of the screen that is inside a window.

Any number of windows may be opened and effectively layered on top of each other. Closing a window reactivates the previously opened window or the original screen, if no other windows have been opened.

It is useful, in some circumstances, to be able to close a window and leave its contents behind on the screen - this is one of the many options available with windows. Other options include Headings, Footings, borders and other effects.

Windows AiF Sequence

ESC [= Y1 ; X1 ; Y2 ; X2 ; WT ; A1 ; ... ; An u

Where:

Y1	Top left-hand row.
X1	Top left-hand column.
Y2	Bottom right-hand row.
X2	Bottom right-hand column.
WT	Describes the window type:
0	No border.
1	Single line border.
2	Double line border.
3	Single line at top and bottom, double at sides.
4	Double line at top and bottom, single at sides.
32	Do not clear screen behind window.
64	Shadow window.
128	Explode window.

A1 .. An Optional parameters to set the window colour. If not present, the current colour Attribute is used.

U Is the literal u.

To open a window starting at the top left-hand corner of the screen, set X1 and Y1 to 1.

To centre the window within the current screen, set both X1 and Y1 to 0 (zero). In this case, the window size is determined by the absolute values of X2 and Y2.

To centre the window in the "zeroed" plane, set either only X1 or Y1 to 0 (zero), i.e. either horizontally or vertically.

Note: the last three values for **WT** are additive, e.g. a single line bordered window that is exploded and shadowed has a **WT** value of 193 (1 + 64 + 128 = 193).

Closing a Window

To close the window use the following sequence:

ESC [= F v

Where:

F = 0 If F is zero or absent then the screen behind the window is restored.

F = 1 To leave window contents on screen.

Example

The application needs to display an important message, and ensure that the user has noticed it by waiting for acknowledgement. Conventional applications reserve one line for messages or try to attract the user's attention by putting the message in reverse video and possibly in some sort of box. However, the user might miss the message, and the message is displayed at the cost and time of having to redisplay the whole of the underlying screen. Use AiF to avoid these problems. To output a message **ERROR!** in White text in a Red window and then wait for user input, use the following AiF escape sequence:

```
ESC[=16;36;18;44;1;0;1;37;41u
ESC[2;2H
ERROR!
input dummy
ESC[=v
```

Window Headings And Footings

A heading and/or footing may only be displayed within the border of a window. If the window has been opened without a border, then headings and footings are ignored.

To put the title in the top of the window border, use the following sequence:

```
ESC _ X1 ; A1 ; ... ; An W .....Title.... ESC \
```

To put the title in the bottom of the window border, use the following sequence:

```
ESC _ X1 ; A1 ; ... ; An U .....Title.... ESC \
```

Where:

X1	Is the starting column of title . If set to 0, or absent, the text is centred.
A1 ; ... ; An	Are the colour attributes.
W	Is the literal 'W' .
U	Is the literal 'U'.
Title	Is the text for the title.
ESC\	Is the terminator.

If the title text is wider than the window border it will be ignored. If no colour attributes are specified then the current window attributes are used.

Footings are treated in the same way as headings with regard to positioning and attributes.

To clear a heading or footing that has been previously placed on a window simply send the heading or footing sequence as appropriate, with the "Title" set to null.

Example

Use the following sequences to open a window with co-ordinates 8,30 and 12,50 in Yellow text on a Blue background with a footing in White text on Red containing the text 'Help'. The window is to be exploded, shadowed and with a border.

```
ESC [= 8 ; 30 ; 12 ; 50 ; 196 ; 0 ; 44 ; 33 ; lu
ESC_0 ; 37 ; 1 ; 41 UHelp ESC\
```

Using AiF menus

There are three AiF sequences that control the use of AiF Menus from within an application.

One sequence enables an application to *load* Menus into the PC memory in readiness for *activation* by the application using a second AiF sequence.

Once a menu has been activated, HostAccess does all the work in processing the user selection and returns the user's choice to the host application. The application can then translate this choice into the required action and process the user's choice as it would normally do. Once an application has received the user's menu choice, it is normal to "close" the menu so that the underlying screen can be updated by the application, if required. Each AiF menu type has its own close menu sequence.

A separate AiF sequence is provided to *clear* specified menus from HostAccess's memory.

The following areas are covered in this section:

- AiF menu types, see below.
- Menu options, see page 118.
- Menu sets, see page 118.
- Colour configuring menus, see page 119.
- Configuring selection characters and separators, see page 120.
- Menu - Exit keys, see page 120.
- Pop-down menus, see page 122.
- Cascading pop-down menus, 127.

AiF Menu Types

Two basic types of menus are supported by AiF - these are pop-down menus and selection boxes.

Pop-down menus work on the principle of a menu bar (usually across the top of the screen) from which lists of menu elements pop-down as the user moves left or right between menu headings on the bar. Any element within a pop-down menu may cascade into another pop-down menu.

Selection boxes (**pop-up menus**) work on the principle of popping up a single list of selections (anywhere on the screen) and allowing the user to move up and down this list to make a choice. To implement the simplest form of AiF Pop-Down menus, see Pop-Down Menus on page 122.

Menu Options

A variety of options is available for AiF menu types. The options include:

Menu Type	Option Description
all	To allow host applications to have access to more than one menu set at any time by saving these into areas of PC memory (called menu slots) without the need to reload each menu set.
all	To colour configure every aspect of any menu.
all	To specify each menu heading's and menu element's selection character for fast single keystroke selection.
all	To provide a variety of exit keys with which the user may choose to leave the menu (e.g. Allowing f10 as a "help exit key" from a menu).
all	To cater for user's typeahead within menu processing.
all	To use a variety of menu styles, including Novell and separator lines within lists of menu elements.
all	To optionally leave the menus displayed on the screen after the user has made a selection.
all	To store menu loading sequences (together with headings and elements) on the PC's disk in special AiF files called forms. This can be useful to speed up menu loading - for more information, see FORMs on page 151.
pop-down	To reposition the pop-down menu bar to any row on the screen.
selection boxes	To run selection boxes (pop-up) menus within AiF windows.

The options that apply to all menu types are documented in the following sections. Menu type specific options are then documented within each menu type in the appropriate Pop-Down menu section.

Local processing of user's keyboard inputs may be controlled by host applications if they need a typeahead facility. Please see Typeahead Mode on page 177 for more details.

Menu sets

Menu sets enable host applications to load more than one menu into the PC's memory at the same time. Each menu set can be instantly activated by sending the appropriate AiF sequence to activate the menu from a specified menu set.

Host applications need to specify the menu set required when they load EACH menu to the PC. This menu set number is the 'SN' parameter (see Pop-Down Menus on page 122) on the load menu sequences and will default to 1 if not specified.

Host applications need to ensure that they do not create menu set number conflicts, by loading two or more menu sets into the same menu set number. In this case, the last loaded menu will be the one used when the menu set number is selected.

HostAccess supports a maximum of 50 menu sets for EACH menu type.

The following overall limitations apply:

Menu Type	Overall Limitations	
Pop-down	Maximum number of pop-down menu sets	50
	Maximum menus per menu set	100
	Maximum menu elements per menu	20
	Maximum menu elements per menu set	2,000
	Maximum number of menu elements across all pop-down menu sets	100,000
Selection box (pop-up)	Maximum number of selection box sets	50
	Maximum size (bytes) of all menu elements per menu set	32,000
	Assuming 30 bytes per menu element, approximate maximum number of menu elements per menu set	1,066
	Approximate maximum number of menu elements across all menu sets	53,300

These limitations are intended as guidelines only - host applications should never need to even approach them. These limitations are also constrained by the available memory on the user's PC.

Colour Configuring Menus

An application using the AiF menus can change the default colours of its menus so that they can be differentiated from other applications' menus. This is done through an extension to the AiF colours sequence. By including a code in the range 97 to 99 as the first parameter, HostAccess applies the subsequent parameter values as colour attributes for the AiF Menus.

These parameters have the following meaning:

- 97** Set the colours of the menu bar and pop-down/up menus.
- 98** Set the colours of the select character.
- 99** Set the colours of the highlighted selection bar.

Both foreground and background colours may be set with each of these parameters.

Note that a menu's colours are determined when the menus are loaded to the PC and are retained with that menu. Menu colours cannot be dynamically changed by an application unless the menu is reloaded after the menu colours have been changed.

Sending these AiF colour sequences with no colour parameters after the 97, 98 or 99 sets the appropriate menu colours back to the system defaults.

This is described in more detail in Using Colours on page 110.

Configuring Selection Characters & Separators

As each menu's list of selections is loaded it is possible to tell HostAccess which character within each element should be highlighted as the selection character.

Manoeuvring through menus by the user is extremely simple and fast and is consistent with the way in which a user would move through HostAccess's own configuration menus .

The first character of each element will be used as the selection character by default, if a particular character is not specified.

To designate any character within the menu element as the selection character, simply prefix the desired character with an ampersand "&". If an ampersand is required as part of the text of a menu element then a double ampersand "&&" can be used to achieve this.

Selection characters may be specified in all menu types and in any menu element (including the menu headings on the menu bar for Pop-Down menus).

If a null menu element is sent to HostAccess, it is treated as a separator. HostAccess displays a line in the position of the null element. This is useful for breaking up groups of different menu choices within a single menu list.

Menu - Exit Keys

All the AiF menus allow the user to exit the menu by pressing an exit key defined by the host application. If these are not defined, a user can exit menu with a carriage return (to show acceptance of the selected elements) or by pressing the ESCape key (to show non-selection of any element and exit the menus completely). However, configurable exit keys give host applications immense flexibility in the way in which they handle user selections within AiF menus.

For example, using defined exit keys, an application can provide a "help" hot-key (say Function Key 10) for any element within any menu. The user is able to press F10 to ask the application to display help and then return instantly back into the menus to move or make a selection.

In addition, exit keys can be used by applications as "menu-wide" exits. For example, you could define F2 to always process the same event, regardless of where the user is within the current menu set. This allows fast manoeuvring and selections of actions from within menus.

Loading Application Specific Exit Keys

To load application specific exit keys as required, use the AiF sequence below:

**ESC _ Z exit_keys ESC **

Where:

- Z** Is the capital letter Z - AiF code for exit keys.
- exit_keys** Are the exit keys that the application will recognise when returned from the AiF menu. These are in the mnemonic format as described in DOS Keyboard Stacker on page 184, e.g. as CR for carriage return, ES for ESCape, F1 for function key 1, etc.

User Response

When exit keys have been configured and loaded to HostAccess the user's response from a menu will be returned to the host application in the following format:

<STX> exit_key <CR> menu_path <CR>

Where:

- <STX>** Is the special start of text character with ASCII decimal value 002.
- exit_key** Is the mnemonic for the exit key used to leave the menus. (as described above). If you define single characters as exit keys, these are returned as 2 characters. For example, 'x' is returned as a space followed by the letter x (i.e. ' X').
- <CR>** Is a carriage return character with ASCII decimal value 013.
- menu_path** Is the path in the form of menu element number(s) for the currently highlighted menu element, delimited where appropriate by commas ",".

Note that the **menu_path** is always returned, regardless of which exit key was used. To reset the exit keys to the default, set **exit-key** to null.

Examples

To load HostAccess with the exit keys required for Menus so that only Carriage Return, ESCape and the Function Keys F1 and F2 are permitted, use the following AiF sequence:

**ESC _ Z F1 F2 ESC **

For example, the menu bar currently highlights the second element in the third pop-down menu. If the user pressed Function Key 2 the following would be returned to the host application:

```
<STX> F2 <CR> 3,2 <CR>
```

Function Keys 1 to 10 mnemonics are F1..F9 with F0 for F10.

If the user pressed Enter the following would be returned to the host application:

```
<STX> CR <CR> 3,2 <CR>
```

Where the first CR is the literal letters "CR" (the other <CR>s are carriage returns, ASCII 013).

It is important to note that exiting either pop-down or selection boxes by pressing the ESCape key will return a menu path of 0,0 or 0 respectively.

Terminal Echo - AiF Menus

You should turn terminal echo off before getting the response from AiF menus, so that this is not displayed on the user's screen. You can use an AiF sequence to suppress host echoed output that may be used instead of the host system's equivalent command. For more details, see Host Echo On/Off on page 156. Terminal echo should be turned back on once the menu response has been input.

Exit keys are common between the AiF Menus, Field Inputs and responses from Image displays - it is the application's responsibility to maintain different exit keys between menus, field inputs and Image displays, if required.

Pop-Down Menu

The AiF sequences needed to use pop-down menus are documented below in the logical order that they would be used by a host application:

Clear pop-down menus.	ESC [= 2 ; SN I
Load pop-down menus.	ESC _ N1 ; SN M H1 ; E1 ; .. ; En ESC \
Activate pop-down menus.	ESC [= 22 ; SN h
Close pop-down menus.	ESC [= 22 I
Get user response to pop-down menus.	See example on page 126.

All the sequences below are assumed to be for menus running in the default AiF menu set, i.e. menu set 1 (one).

Clearing Menus

The following AiF sequence clears all application Pop-Down menus from the menu set SN in the PC's memory:

```
ESC [= 2 ; SN I
```

Where SN is an integer between 1 to 50.

Loading Menus

To load a pop-down menu into HostAccess's memory on the PC, use the following sequence.

```
ESC _ N1 ; SN M H1 ; E1 ; .. ; En ESC \
```

Where:

- N1** Is the menu number from 1 to 100. Only menu numbers 1 to 8 may be presented from the menu bar. The first element in each of these menus will be taken as the heading for that pop-down menu.
- SN** Is the menu set number from 1 to 50 (if not specified, default is one).
- M** Is the capital letter 'M' - AiF code for Menus.
- H1** Is the first parameter and will be used as this Pop-Down Menu's heading in the Menu Bar for menu numbers 1 to 8 only. For all other menus, this is the first menu element.

E1..En Are the menu elements, up to 20 per Pop-Down Menu.

Note: Menu Heading and Element text strings should not contain semi-colons (interpreted as element delimiters). Nor should they contain control characters, which will corrupt the menu text.

Using Pop-down Menus

Up to 100 menus may be loaded at any one time and each menu may contain up to 20 elements. Each element may be a maximum of 78 characters long. This means that one menu set can be used to present the user with up to 2,000 options.

Pop-down menus automatically justify menu headings across the menu bar. A maximum of 8 menu headings may be presented on the menu bar. Menu headings are truncated if their total width exceeds the current screen width.

Menus can be dynamically reconfigured to suit the application's requirements. At any point, the host application may reload any menu simply by re-sending the AiF sequence for that particular menu. In this way, menus can be modified dependent upon how the application interprets the user's selections, actions, access security and so on.

Activating Pop-Down Menus

Use the following sequences to activate the Pop-Down Menus for user interaction:

ESC [= 22 ; SN h

Runs the menu set number SN starting at menu 1 element 1, unless the menu has been used before, in which case it starts at the last selection.

You may optionally extend this sequence to specify that a specific element in a specific menu be activated as the highlighted option when the menu set is invoked by the AiF sequence below:

ESC [= 22 ; SN ; m ; e ; Y1 h

Where:

- SN** Is the menu set number and is integer between 1 and 50.
- m** Is the menu number to be activated.
- e** Is the menu element number to be activated.
- Y1** Is the row number on which the menu bar will be displayed (i.e. between 1 and the current screen depth).

If a row number that equals the current screen depth (normally 24) is used then no pop-down menus will be available from the menu bar.

This sequence will run the current menu set number **SN** starting at top-level menu 'm', element 'e'.

If the **m;e** parameters are set to 0;0 then HostAccess automatically activates the menu at the last 'remembered' menu position and redisplay the menu set through to the last selected option (within cascades if appropriate). If these are set to "0;0" and this is the first time the menu is invoked, then the option in menu 1, element 1 is highlighted when the menu is activated.

When the Pop-Down Menus are activated they appear over the existing screen. As the user moves through the menus, HostAccess instantly refreshes the underlying screen.

Getting a Pop-Down Menu Response

AiF Pop-Down Menus allow a user to manoeuvre around the menus using the up, down, left and right arrow keys. The Home and End keys will move the highlight bar to the top and bottom of a Pop-Down Menu list. To select a menu option, the user presses the Enter key (or a valid exit key) when the highlighted element is the required option. Pressing the ESCape key at any point allows the user to exit from the Pop-Down Menus. To jump left or right between menus when on the menu bar, press <Control> plus the letter of the menu's highlighted selection character.

HostAccess's AiF handles all of the menu movement and underlying screen refreshes without any intervention or additional code on the host system. Once a user selects an option, AiF will send the selected Menu number and the Element number to the Host system. It is a simple matter for the host application to interrogate this response and determine which option the user has chosen.

Pop-Down Menu Response

The Pop-Down Menu response from AiF takes the format:

<STX> exit_key <CR> Mn,En <CR>

Where:

<STX> Is a special Start of Text character (ASCII decimal value 002). This character indicates the start of the menu response string and enables the application to discard any typeahead input that could have been sent before the response string itself.

exit_key Is the Exit Key mnemonic in the AiF Keyboard Stacker format as defined on DOS Keyboard Stacker (page 184) - also see notes on Loading Exit Keys (page 139) for the key used by the user to exit from the menu.

<CR> Is a carriage return (ASCII decimal value 013).

Mn,En Is the menu path of the option selected as returned by HostAccess, Where:

Mn Is the number of the option selected, from 1 to 100.

En Is the menu element number of the option selected digit number, from 01 to 20.

Please refer to the examples below to see the logic required to obtain an AiF menu response.

If the user hits the ESCape key whilst in menus, the menu number and element number are both returned with a value of zero.

Closing Pop-Down Menus

Use the following sequences to close the Pop-Down Menu:

ESC [= 22 I

closes the currently open Pop-Down Menu.

When the Pop-Down Menus are activated they appear over the existing screen. If another Pop-Down Menu had previously been activated, it would have been closed by the activation of the current menu. In other words, there is no concept of a "stack" of open Pop-Down Menus and applications should only need to issue one close menu command.

A close menu sequence must be sent to HostAccess before the application can correctly update the underlying screen. If the menu is not closed, the screen update may appear within the "window" opened for the currently active Pop-Down Menu's element list.

Closing the pop-down menu does not remove it from the PC's memory. The same menu may be reactivated at any time with the "activate menu" AiF sequence.

Pop-down menu example 1

To change the application menus to be Black letters on a Light gray background, with the select character in Red and the selection bar with White characters on a Black background use the following 'code':

```
menu_color = 'ESC[=97;0;'
select_color = 'ESC[=98;0;'
hilight_color = 'ESC[=99;0;'
send to PC menu_color '30;47m'
send to PC select_color '31;47m'
send to PC hilight_color '1;37;40m'
```

Pop-down menu example 2

The code below illustrates a schematic structure for loading and activating AiF's Pop-Down Menus and for interrogating the AiF menu response. Developers can use this type of logic to provide their applications with a common routine that handles all the AiF menus processing.

```
* assign variables for menu text and number of menus *
get menu_text
get nbr_of_menus
* clear any existing menus (from menu set 1) *
send to PC 'ESC[=2;1l'
* load the Pop-Down Menu using the following code (into the *
* default menu set number 1) *
menu_nbr = 1
loop
while menu_nbr <= nbr_of_menus
  this_menu = menu_nbr ; 1 'M' menu_text(menu_nbr)
  send to PC 'ESC_U this_menu 'ESC\'
  menu_nbr = menu_nbr + 1
repeat
* activate the pop-down menu *
* and interrogate the AiF Menu response as follows *
quit = false
loop
  * get AiF menu response prefix *
  send to PC 'ESC[=22;1h'
  loop
    turn terminal echo off
    input menu_prefix
```

```

        until ASCII char 002 found in menu_prefix do
        repeat
* now get the user's menu selection *
        menu_nbr = 0
        input menu_coordinates
        extract menu_nbr (from menu_coordinates)
        extract menu_element_nbr (from menu_coordinates)
        if menu_nbr = '0' then
            quit = true
        end if
        turn terminal echo on
* now process the option selected... *
* as any application would do so ... *
* Remember, you will need to close the menu if
* you need to update the screen ... *
until quit do repeat

```

This 'code' loops around the host system's input buffer waiting for a valid AiF start of menu response character. This is necessary to discard any redundant input in the input buffer.

Once this is found, the menu option selected may be easily determined by extracting the menu number and the element number from the next input. If the user has hit the ESCape key this is detected (menu_nbr = 0) and the process exits the loop. Otherwise the menu option chosen is processed normally, as any application would do.

Notes: You should load pop-down menus into (and cleared from) specific menu slots, to avoid clashes with other application's AiF menus - host applications will need to manage menu slot numbers to prevent conflicts.

You may wish to use the AiF sequence to push and pop complete "environments" before loading/reloading application specific menu sets. This will ensure that there is no possibility of their menu sets clashing with another application. Please see Save Environment on page 147 for more information.

Cascading Pop-Down Menus

Cascading menus may be used to enhance existing Pop-Down menus by giving applications the ability to control and display a menu tree structure. The use and operation of cascading menus is simply an extension to HostAccess's existing Pop-Down menu system.

The appropriate AiF sequences for cascading pop-down menus are described in the next sections.

Resetting Cascading Menus

The following AiF sequence resets menu i.e. clears the menu set in the menu set number SN from the PC's memory.

```
ESC [= 2 ; SN I
```

Where:

SN Is the menu set number from 1 to 50.

This sequence need only be used if the application wishes to reload a completely new set of cascading menus into the same menu set number.

Loading Cascading Pop-Down Menus

To load a cascading pop-down menu into HostAccess's memory on the PC use the following sequence:

```
ESC _ N1 ; SN M H1 ; E1 ; E2 TC CN ; .. ; En ESC \
```

Where:

N1 Is the menu number from 1 to 100. Only menu numbers 1 to 8 may be presented from the menu bar. The first element in these each of these menus will be taken as the heading for that pop-down menu.

SN Is the Menu Set number from 1 to 50.

M Is the capital letter 'M' - AiF code for Menus.

H1 Is the first parameter and will be used as this Pop-Down Menu's heading in the Menu Bar for menu numbers 1 to 8 only. For all other menus, this is the first menu element.

E1..En Are the menu elements, up to 20 per Pop-Down Menu.

TC Is the cascade tag character indicating that if this element is selected by the user a cascaded menu will be opened. The default tag character is a split vertical bar (|).

CN Is the menu number of the cascaded menu. This menu should itself be loaded with this menu number.

Note: menu heading and element text strings should not contain semi-colons (interpreted as element delimiters). Nor should they contain control characters which will corrupt the menu text. Each menu may be loaded individually by addressing it by its menu number. In other words, menus can be dynamically reconfigured to suit the application's requirements.

Example

Cascaded menus are "pointed to" within the text of any menu element by suffixing the text with a cascade tag character followed by the number of the menu to be cascaded to. For example:

```
ESC _ 1 ; 1 M First Menu ; 1st Element ; 2nd Element | 16 ESC \
```

This AiF sequence will load a menu into HostAccess as menu number 1 in menu set number 1 with a menu heading of "First Menu" and with two menu elements "1st" and "2nd". The second element, when selected by the user, will cascade into menu number 16 allowing the user to select a further menu element from this cascaded menu. Needless to say, menu elements in this cascaded menu may themselves cascade into other menus, and so on.

Activating Cascading Menus

Use the following AiF sequence to activate the Pop-Down Menus for user interaction:

```
ESC [= 22 ; SN ; m ; e ; Y1 h
```

Where:

SN Is the menu set number.

m ; e Is the path to a top-level menu or the last selected menu element.

Y1 Is the row number on which the menu bar will be displayed (i.e. between 1 and the current screen depth). Defaults to 1.

This sequence will run the current menu set number **SN** starting at top-level menu 'm', element 'e'.

If the "m ; e" parameters are set to "0;0" then HostAccess automatically activates the menu at the last 'remembered' menu position and redisplay the menu set through to the last selected option (within cascades if appropriate). If these are set to "0;0" and this is the first time the menu is invoked, then the option in menu 1, element 1 is highlighted when the menu is activated.

When the cascading menus are activated they appear over the existing screen. As the user moves through the menus, HostAccess refreshes the underlying screen instantaneously.

Getting a Menu Response

AiF Pop-Down Menus allow a user to move through them using the up, down, left and right arrow keys. The Home and End keys will move the highlight bar to the top and bottom of a Pop-Down Menu list. To select a menu option, the user presses the Enter key when the highlighted element is the required option. Pressing the ESCape key at any point allows the user to exit from the Pop-Down Menu.

HostAccess's AiF handles all of the menu movement and underlying screen refreshes without any intervention or additional code on the host system. Once a user selects an option, AiF will send the full menu path for the selected menu number and the element number to the host system. It is a simple matter for the host application to interrogate this response and determine which option the user has chosen.

Cascading Menu Response Format

The Cascading Menu response from AiF takes the format:

<STX> exit_key <CR> Mn,En .. Mn,En <CR>

Where:

- <STX>** Is a special Start of Text character (ASCII decimal value 002). This character indicates the start of the menu response string and enables the application to discard any typeahead input that could have been sent before the response string itself.
- exit_key** Is the Exit Key mnemonic in the AiF Keyboard Stacker format (as defined on DOS Keyboard Stacker- also see notes on Loading Exit Keys).
- <CR>** Is a carriage return (ASCII decimal value 013).
- Mn** Is the number of the option selected and is a single digit number from 1 to 100.
- En** Is the menu element number of the option selected and is a two digit number from 01 to 20.

The full menu path for cascaded menus is returned in the form of pairs of menu and element numbers. Each pair of numbers is separated by a comma.

If the user hits the ESCape key while in the menus, the menu number and element number are both returned with a value of zero.

Closing Pop-Down Cascading Menus

Use the following sequences to close the Pop-Down Menu:

ESC [= 22 I

closes the currently open Pop-Down Menu.

When the Pop-Down Menus are activated they appear over the existing screen. If another Pop-Down Menu had previously been activated, it would have been closed by the activation of the current menu. In other words, there is no concept of a "stack" of open Pop-Down Menus and applications should only need to issue one close menu command.

A close menu sequence must be sent to HostAccess before the application can correctly update the underlying screen. If the menu is not closed, the screen update may appear within the "window" opened for the currently active Pop-Down Menu's element list.

Note: A simple form of horizontally scrolling menus may be simulated by using the row number in combination with menu number 1 to 8 without any pop-down menu elements.

Using Selection Boxes

Selection boxes provide another menu system within HostAccess's AiF. They give applications the ability to present the user with a scrolling window over a list of menu elements. The user may select any element within this list, as well as being able to rapidly scroll or page up/down this list. A Selection Box may be "popped-up" at any screen position and may optionally appear over or alongside other Selection Boxes or pop-down menus, open windows, and so on.

The AiF sequences that can be used to control Selection Boxes are detailed in the following sections.

Resetting Selection Boxes

The following AiF sequence clears the Selection Box set number **SN** (1 to 50) from the PC's memory.

ESC [= 24 ; SN I

This sequence need only be used if the application wishes to reload a completely new Selection Box into the same selection box set number.

Loading Selection Boxes

To load a Selection Box into HostAccess's memory on the PC use the following sequence:

ESC_SN ; Y1 ; X1 ; DT ; BT ; MT ; MW S H1 ; E1 ; E2..En ESC

Where:

- SN** Is the selection box set number from 1 to 50.
- Y1** Is the start row for display.
- X1** Is the start column for display. If both x1 and y1 are set to 0 (zero) then the selection box is centred on the screen if no other selection box, pop-down menu or window is open ; otherwise, the selection box is positioned to the right of the menu or window.
- DT** Is the depth of the Pop-Up menu (number of rows in the box). Defaults to the number of elements in the selection list, or 20 if there are more than 20 elements.

- BT** Describes the box type:
- 0 : No border.
 - 1 : Single line border.
 - 2 : Double line border.
 - 3 : Single line at top and bottom, double at sides.
 - 4 : Double line at top and bottom, single at sides.
 - 32 : Do not clear screen behind menu box.
 - 64 : Shadow menu box.
 - 128 : Explode menu box

The last three values for BT are additive, e.g. a single line bordered menu box that is exploded and shadowed has a BT value of 193 (1 + 64 + 128).

- MT** Describes the menu type as: normal (0) or Novell menu style (1)
- MW** Is the maximum width of the Selection Box. Defaults to the width of the longest element.
- S** Is the letter 'S', AiF code for Pop-Up menu (Selection Box).
- H1** Is the Selection Box heading. If no heading is required this should be null.
- E1..En** Are menu elements, separated by semi-colons';'.

HostAccess allocates up to 32K of PC memory for Selection Box elements within each menu set. If the average length of each element were 40 characters, this memory could contain over 800 elements. Applications developers should be cautioned against presenting the user with large numbers of elements. Users needing to select element 622 will not enjoy scrolling through the preceding 621 elements nor waiting for this list to be sent to the PC over a communications link!

Activating Selection Boxes

There are two AiF sequences which can be used to activate selection boxes.

Use the following sequence to activate a selection box for user interaction within a window automatically sized by AiF:

ESC [= 23 ; SN ; En h

Where:

SN Is the selection box set number.

En Is the start menu element number. If set to 0, the menu element highlighted is that last selected by the user, or menu element number 1 if this is the first time the selection box has been activated.

This sequence tells AiF to size the width of the selection box automatically based upon the longest menu element or using the DT and MW parameters as specified in the selection box load sequence. The selection box is cleared and the underlying screen instantaneously restored, once the user has made a selection and exited the selection box.

To activate a selection box within a previously opened window use the following sequence:

ESC [= 24 ; SN ; En h

Where:

SN and En Are as defined above.

When using this sequence, it is the host application's responsibility to close (and optionally clear) the window within which the Selection Box is activated. This activation sequence is also useful when an application needs to make a Selection Box appear on the screen within a window with a heading and/or footing, and where the window is used to determine the menu's width regardless of the maximum menu element width.

Getting a Selection Box Response

AiF Selection Boxes allow a user to scroll up and down the menu using the up and down arrow keys. The Home and End keys move the highlight bar to the top and bottom of a Pop-Up Menu. To select a menu element, the user presses the Enter key (or a valid exit key) when the element is highlighted. Pressing the ESCape key at any point allows the user to exit from the Selection Boxes.

Once a user makes a selection, AiF sends the selected Element number to the host system. It is a simple matter for the host application to interrogate this response and determine which element the user has chosen.

The Selection Box response from AiF takes the format:

<STX> exit_key <CR> En <CR>

Where:

- <STX>** Is a special Start of Text character (ASCII decimal value 002).
- exit_key** Is the Exit Key mnemonic in the Keyboard Stacker format (see DOS Keyboard Stacker on page 184 and notes on Loading Exit Keys on page 139).
- <CR>** Is a carriage return (ASCII decimal value 013).
- En** Is the number of the menu element selected.

If the user hits the ESCape key to exit from the selection box, the element number is returned with a value of zero.

Closing Selection Boxes

Use the following sequences to close the selection box:

ESC [= 23 I

Closes the currently open selection box.

When the selection boxes are activated they appear over the existing screen. If another selection box had previously been activated, it would NOT have been closed by the activation of the current selection box.

In other words, there is the concept of a "stack" of open selection boxes and applications can have any number of selection boxes displayed on the screen at any time. Obviously, applications should issue a close selection box command to close each of the opened selection boxes.

Note: A close selection box sequence must be sent to HostAccess before the application can correctly update the underlying screens. If the selection box is not closed, the screen update may appear within the "window" opened for the currently active selection box's element list. In addition, an open selection box must be closed before being re-loaded with a new list of elements.

Example

The following pseudo-code will open a Selection Box (in selection box set 1, after first clearing it) starting at column 10, row 5, 3 rows deep and return to the host the number of the element selected from a list of 6 choices. The Selection Box would also be framed with a single line border, shadowed and have a heading of "Choose Option".

```
options_list = 'Choose Option;1st option;2nd Option;3rd
Option;4th Option;5th Option;6th Option'
send to PC ESC '[=24;11'
send to PC ESC '_1;5;10;3;65;0S' options_list ESC '\
send to PC ESC '[=23;1;1h'
loop until STX (ASCII dec 002) found in user_input do repeat
get menu_choice
if menu_choice equals 1 then
send to PC menu_choice 'st option selected'
if menu_choice equals 2 then
send to PC menu_choice 'nd option selected'
etc. to process each choice, as appropriate.
```

No consideration has been given to loading and interrogating which Exit Key was used by the user to exit the Selection Box, nor to colour configuration and so on.

Notes: Positioning of the Selection Box can be automatically adjusted to take into account an active Menu (either Pop-Down or another Selection Box) by specifying special values for the Y1 and X1 parameters in the Load Selection Box AiF sequence. If these parameters are set to 0 (zero), HostAccess will attempt to "best" position the Selection Box to the currently selected option in the active (last displayed) Menu.

If either of these parameters is 0 and there is no other menu active, positioning is centred on either the column or row (or both) in the currently open Window (or screen). Using this feature it is possible effectively to cascade Selection Boxes from Pop-Down menus.

Using Field Input

Processing user keyboard inputs can slow down host applications. This can be aggravated by the need for the host to **echo** each key entered by the user, which can also strain the capacity of networks as each character is transmitted within its own "network packet".

Users accessing the host through public service networks at slower line speeds often suffer longer response times because of this need to echo each character. Developers are burdened with having to write complicated, single character, terminal type dependent input routines to provide users with any form of editing other than backspace.

HostAccess's AiF Field Input facilities change all of this.

You can now implement a fast, sophisticated field input system within applications - whilst still being able to maintain complete control from the host application. Field input can be optimised to use the PC's processing power with generally no more a couple of lines of code.

How AiF Field Input Works

There are basically three AiF sequences that can control user's input from within a host application.

Each AiF sequence simply informs HostAccess that the user is about to start input. At this point control is passed to HostAccess. The user then inputs one or more keystrokes and has access to all of the standard DOS-like facilities for amending, inserting, deleting and moving through the input text.

When the user presses a valid exit key, the contents of the user's input are sent back to the host.

Field Input Types

Three types of field inputs are supported by AiF:

Line input is the simplest field input type and works on the principle of "cutting" a specified number of characters out of the screen from a specified cursor position.

Box input enables user inputs to take place within a fixed width single line box and for the box to be enhanced with box attributes (such as frames, colour).

Window Editor gives applications the ability to define a variable width window over one or more input rows and for the window to be enhanced with window attributes (frame, colour, titles). This effectively gives users an on-screen "mini" word processor.

Each of these types is documented in the following sections.

Local processing of user's keyboard inputs may be controlled by host applications if they need a typeahead facility. Please see Typeahead Mode, see page 177 for more details.

User Keys Available for Field Input

The keys available for the user to move around and modify input text are the same for all field input types. Users may move around text by using the arrow keys. Overwrite mode is indicated by a solid

(block) cursor. Insert mode is indicated by an underline cursor. Pressing the Insert <INS> key will toggle between insert and overwrite modes.

The contents of the input field are always returned to the host application. It is the host application's responsibility to determine if the field has been changed and how the user finished (exited) the field input. HostAccess returns to the host the field's contents and an exit key. If exit keys are loaded, you can use these to exit the field input and return to the application control.

Otherwise, the user may press the ESCape key to exit without changing the field's contents. In Line and Box input types, carriage return or ESCape exits field input and sends the text to the host application. If the user is within the Window Editor field input type, then pressing F1 exits field input and sends the text back to the host application.

If the user is within Line or Box input and starts to key input without moving along the text, then the text displayed is cleared and the user's input completely replaces the old field. This facility is optional and may be controlled by the host application.

A number of local editing keys are also available for use while within field input. These may be Windows, WordStar or WordPerfect compatible, depending upon which mode the user has configured via the Editing... option of the Configure menu.

Field Input - Exit Keys

All the AiF Field Inputs allow users to exit input by pressing a host application defined exit key.

If these exit keys are not defined, a user can indicate acceptance of the input by entering a carriage return (but note that within Window Editor the F1 key is the default "acceptance" exit). The ESCape key may be pressed to exit and indicate non-acceptance of the input. However, configurable exit keys give host applications immense flexibility in the way in which they handle user inputs.

For example, using defined exit keys it is possible for an application to provide a "help" hot-key (say Function Key 10) for any field input anywhere in an application. The user is able to press F10 to ask the application to display help and then return immediately to the field input to continue entering data.

In addition, exit keys can be also used by applications as "input field-wide" exits. For example, pressing F2 could always cause the application to process the same event, regardless of where the user is within the current field input. This provides for very fast manoeuvring and selections of actions once the user is familiar with the exit keys defined by the application.

Loading Exit Keys

Host applications may load application specific exit keys as required by using the AiF sequence below.

```
ESC _ Z exit_keys ESC \
```

Where:

- Z** Is the capital letter Z - AiF code for exit keys.
- exit_keys** Are the exit keys that the application will recognise when returned from the AiF field input. These are as defined in DOS Keyboard Stacker on page 184.

Field Input Response

The user's response is returned to the host application in the following format:

```
<STX> exit_key <CR> input_str <CR>
```

Where:

- <STX>** Is the special start of text character with ASCII decimal value 002.
- exit_key** Is a two character alphanumeric mnemonic for the exit key used to leave the field input. The mnemonics returned are the same as those used when loading the exit keys.

If you specify single characters as exit keys, these will be returned as two characters. For example, if 'X' is specified as an exit key, it will returned as a space followed by the letter X (i.e. ' X').

- <CR>** Is a carriage return character with ASCII decimal value 013.
- input_str** Is the string as input by the user. Note that the response from Window Editor is expanded to return each line of the window as a separate input - please see Window Editor on page 144.

Example

To load HostAccess with the exit keys required for Field Input so that only Carriage Return, ESCape and the Function Keys F1 and F2 are permitted, use the following AiF sequence:

```
ESC _ Z F1 F2 ESC \
```

Terminal Echo

You should turn terminal echo off before getting the response from AiF Field Input, so that this is not displayed on the user's screen. You can use an AiF sequence to suppress host echoed output, instead of the host system's equivalent command. For more details on this please see Host Echo On/Off on page 156. Terminal echo should be turned back on once the menu response has been input.

Exit keys are common between the AiF Menus, Field Inputs and responses from Image displays - it is the application's responsibility to maintain different exit keys between menus, field inputs and Image displays, if required.

Line Input

This sequence enables host applications to define a fixed length field on part or all of one row on the screen for user input. Any text already on the screen within that field will be taken as the initial input text by the user. The user can modify this text before sending it to the host application.

ESC [= 25 ; FL ; VL ; SM h

activates line input at the current cursor position.

Where:

- FL** Is the integer field length (as a number of columns on the screen) from which to "cut" text from the screen. It also defines the maximum number of characters that may be input.
- VL** Is the validation parameter as any one of the following:
- 0 No validation (any input accepted).
 - 1 Integer input only.
 - 2 Numeric input only.
 - 3 Alphabetic input only.
 - 4 Alphanumeric input.
 - 5 Hex input.
 - 6 Hidden input (input is echoed as # characters: useful for entering passwords).
- SM** Is the start mode code for the action to be if the first user input character is not an editing action, i.e. is a character to enter into the field. This parameter should be assigned as:
- 0 No special action for first user input.
 - 1 If the first user input character is not an editing character, clear the field first.

The application must place the cursor at the (column, row) position from which input is to take place immediately BEFORE sending this AiF sequence.

Getting a Line Input response

Line input responses from AiF take the format:

```
<STX> exit_key <CR> input_str <CR>
```

Where:

- <STX>** Is a special Start of Text character (ASCII decimal value 002).
This character indicates the start of the field input response string and enables the application to discard any typeahead input that could have been sent before the response string itself.
- exit_key** Is the Exit Key mnemonic in the AiF Keyboard Stacker format (as defined in DOS Keyboard Stacker on page 184 – also see notes on Loading Exit Keys on page 139).
- <CR>** Is a carriage return (ASCII decimal value 013).
- input_str** Is the text as input by the user.

Application Examples

Applications tend to use the same start column for multiple fields, to display the contents of the fields and then allow the user to modify a field by skipping up and down lines.

As each field is input the application redisplay the contents, permits the user to input new text and then displays the new field again.

With this AiF sequence, there is no need to do this. The application displays the field contents, places the cursor at the start of the field and switches AiF line input on. Redisplay of the field's text is unnecessary as the text the user sees on the screen is what has been sent to the host application.

Note: Trailing spaces are trimmed from the input before being sent back to the host application.

Box Input

Box Input enables host applications to pop up an Input Box anywhere on the screen and request user input. This input may be optionally validated within the PC before being sent back to the host. Validation can be specified to optionally restrict the user's input to numeric only, alphanumeric, hidden (for passwords) and so on.

User input is NOT constrained to the box's width. Text input may exceed the width of the box and AiF will indicate that additional text exists by displaying appropriate arrow symbols. The maximum length for the text that can be input should be specified by the host application.

The Input Box itself may be enhanced with selected frame styles and/or title. Once the user has completed input, the box may optionally disappear and the underlying screen be immediately restored.

Box input provides an especially elegant adjunct to handling user input from AiF pop-up and pop-down menus. HostAccess users will probably already be familiar with the Box Input style as it is used within HostAccess's own configuration menus.

Box AiF Sequence

The following AiF sequence enables you to define an input box.

**ESC _ Y1 ; X1 ; FL ; BS ; BT ; VL ; SM J text ; title ESC **

Where:

- Y1** Is the top left-hand row.
- X1** Is the top left-hand column.
Setting X1 and Y1 to 1 displays a box starting at the top left-hand corner of the screen.
Setting X1 and Y1 to 0 displays a box starting at the current cursor position (or currently selected menu element).
- FL** Is the maximum field input length.
- BS** Is the width of box. This may be less than the field input length. If zero, the maximum field input length is used.
- BT** Describes the box type:
 - 0 No frame.
 - 1 Single line frame.
 - 2 Double line frame.
 - 3 Single line at top and bottom, double at sides.
 - 4 Double line at top and bottom, single at sides.
 - 64 Shadow window.
 - 128 Explode window

The last two values for BT are additive, e.g. a single line framed box that is exploded and shadowed has a BT value of 193 (1 + 64 + 128).

- VL** Is the validation parameter as any one of the following:
- 0 No validation (any input accepted).
 - 1 Integer input only.
 - 2 Numeric input only.
 - 3 Alphabetic input only.
 - 4 Alphanumeric input.
 - 5 Hex input.
 - 6 Hidden input (input is echoed as # characters: useful for entering passwords).
- SM** Is the start mode code for the action to be if the first user input character is not an editing action, i.e. is a character to enter into the field. This parameter should be assigned as:
- 0 No special action for first user input.
 - 1 If the first user input character is not an editing character, clear the field first.
- J** Is the capital letter 'J': AiF code for Input.
- text** Is the input text to be displayed when the Input Box is activated and modified by the user (may be null).
- title** Is the heading text for the Input Box (may be null). If a box type of 0 (no frame) is used, the title is discarded.

Getting a Box Input Response

Box input responses from AiF take the format:

<STX> exit_key <CR> input_str <CR>

Where:

- <STX>** Is a special Start of Text character (ASCII decimal value 002).
This character indicates the start of the field input response string and enables the application to discard any typeahead input that could have been sent before the response string itself.
- exit_key** Is the Exit Key mnemonic in the AiF Keyboard Stacker format (as defined in DOS Keyboard Stacker on page 184 - also see notes on Loading Exit Keys on page 139).
- <CR>** Is a carriage return (ASCII decimal value 013).
- input_str** Is the text as input by the user.

Once the user has exited from the Input Box this box is automatically cleared from the screen and the underlying screen restored immediately.

Box Input Response Examples

To pop-up an Input Box on row 12, column 15 of the screen to prompt the user to enter a 50-character, alphabetic-only surname within a box 25 characters wide, an application should use the following AiF sequences:

```
send to PC 'ESC[=12;15;25;50;1;3;0;J;Enter Surname ESC \'
```

The above Input Box has a single line frame with a heading of "Enter Surname". The maximum input length is 50 characters within a box width of 25. Also note that the box will be cleared from the screen when the user's input is complete.

To prompt the user with an existing surname, say "Harvey" (that can be accepted or amended by the user), simply use the sequence below:

```
send to PC 'ESC[=12;15;25;50;1;3;0;JHarvey;Enter Surname ESC \'
```

To prompt the user for the same surname after, say, a pop-down menu or Selection Box option "change surname" has been selected, use the sequence below:

```
send to PC 'ESC[=0;0;25;50;1;3;0;JHarvey;Enter Surname ESC\'
```

Note: both the Y and X co-ordinates of the box have been set to 0 (zero). This means that HostAccess will position the box at the current cursor co-ordinates. Where this is done from an AiF menu, the current cursor position is taken as being one row below the highlighted element with the start column at the centre of the element. This positioning occurs automatically but may be constrained by factors such as the width of the input box and proximity to the edges of the screen.

Trailing spaces are trimmed from the input before being sent back to the host application. Box input limits the entry to just one line on the screen. Applications requiring multiple lines of input should use the Window Editor AiF sequence described in Window Editor.

Window Editor

The AiF Window Editor enables host applications to pop up an input window (of one or more lines) anywhere on the screen and request user input. The user may manoeuvre around the text within the window using all of the available PC based keys to insert, delete and add text.

User input is constrained by the window's width and depth.

The input window itself may be enhanced with selected frame styles and/or titles in accordance with AiF Windows. In other words, the host application should normally open an AiF window before activating the Window Editor. Likewise, once the user has completed input, an AiF sequence should be used to close and optionally clear the input window.

The Window Editor is particularly suited to applications that need to capture user notes. It, in effect, acts as a very fast local word processor with the major benefit that the user's notes are only sent to the host once the user is satisfied with the text that has been entered. This has considerable performance advantages for applications running over networks or asynchronous lines.

ESC [= 26 h

switches the window editor on, using the text in the currently open window.

Getting A Window Editor Response

Window Editor responses from AiF take the format:

<STX> exit_key <CR> inp_str1 <CR> .. inp_str NN <CR><STX><CR>

Where:

<STX>	Is a special Start of Text character (ASCII decimal value 002). This character indicates the start of the field input response string and enables the application to discard any typeahead input that could have been sent before the response string itself.
exit_key	Is the Exit Key mnemonic in the AiF Keyboard Stacker format (as defined in DOS Keyboard Stacker on page 184 – also see notes on Loading Exit Keys on page 139).
<CR>	Is a carriage return (ASCII decimal value 013).
inp_str1 .. inp_str NN	Is the text input by the user for each line within the input window, where NN is the number of lines in the window.

It is important to note that the response from Window Editor is terminated with another STX character. The advantage of this is that null trailing lines are not returned to the host. Applications should therefore always check for the trailing STX character and use this to identify the end of the user inputs.

Window Editor examples

To pop-up an input window on row 18, column 10 of the screen to prompt the user to enter up to 4 lines of input text with each line not exceeding 35 characters, an application should use the following AiF sequences:

Open window and add title first:

```
send to PC 'ESC[=18;10;21;44;193;0;1;33;41u'
send to PC 'ESC_WInput text WindowESC\'
```

Activate Window Editor:

```
send to PC 'ESC[=26h'
```

Look for the start of text delimiter indicating user has finished input:

```
loop
  get user_response
until 1st character of user_response eq ASCII 002
repeat
```

The exit key should be interrogated at this point:

```
exit_key = 2nd character onwards of user_response
```

Now get the user text from the input window (input this into an array called user_text):

```
ctr = 1
loop
  get user_response
until user response eq ASCII 002
  user_text(ctr) = user_response
  ctr = ctr + 1
do repeat
```

Finally, close the window leaving it visible on the screen:

```
send to PC 'ESC[=1v'
```

Notes: Trailing spaces are trimmed from the input before being sent back to the host application.

If exit keys have not been loaded (see Loading Exit Keys on page 139) then the ESCape key and the Function key F1 are available as exit keys by default.

Save Environment

Host application programs that have been designed to interface with other (possibly unknown) applications which might also be using AiF facilities may experience "conflicts of AiF interest" with the other applications, such as loading AiF menus into the same menu set number, changing but not resetting screen colours and so on.

These conflicts may be avoided by ensuring that applications "save" their AiF environment at an appropriate point by using the AiF sequence in the following sections.

Push environment

ESC [= 99 p

Pop environment

ESC [= 99 q

Push environment will essentially save everything except backpages for the currently active session including:

- The current screen.
- The screen mode (row & column settings).
- AiF windows.
- Screen attributes.
- AiF stacks and slots, menus and selection boxes.
- The screen's background fill character and attribute.

Session dependent attributes saved include:

- Cap lock and num lock status.
- Mouse status.
- Function keys, AiF exit keys and key changes.
- Current emulation in use.

Application Environment Examples

Applications providing gateways out to other applications that might themselves make use of AiF should save their own environments before shelling out to the gateway. Obviously, the application would want to restore its original environment once control has been returned from the gateway.

Notes: You should ensure that the "environments" saved stack is popped in the correct order to properly restore environments for earlier programs. Saving environments can consume sizeable chunks of the PC's memory depending upon the number and size of menu and selection boxes loaded, the number of screens pushed on to the screen slots stack etc.

Display Optimisation

Users are highly sensitive to the time an application takes to paint the screen. If the user is using a dial-up link or a network, it can take several seconds to paint a complex screen. These types of delays are often more critical when a user is stepping through applications screens than when waiting for a response to a complex transaction.

HostAccess's Display Optimisation features address this problem by providing programmable facilities to dramatically speed up screen updating.

The following sections describe how host applications can make use of the AiF features called SLOTS, FORMs and FREEZE ON/OFF. All of these features have been designed to minimise the time, or perception of the time, taken to display host application screens. In many cases, this time can be reduced by more than a hundred-fold.

Overview of Features

SLOTS make use of the PC's memory to save and restore screen images - this is the fastest method of redisplaying an application screen. Screen images may be pushed on to a stack in the PC's memory and popped off the stack as required by the application. Screen display times are virtually instantaneous - the user no longer needs to wait for 2 or more seconds while the host system sends 2,000 or more bytes to redisplay the screen.

FORMs is another means of saving and restoring screen images and display text which are stored on the PC's disk drive instead of within memory.

FREEZE ON/OFF is a very clever technique for improving the user's perception of screen display times. If an application turns FREEZE ON, builds the screen (this is done in the memory of the PC and is not visible to the user) and then turns FREEZE OFF, the effect is one of instant screen display.

SLOTS

AiF SLOTS provide the facility to store any screen image into the memory of the PC. You can address these images as 'SLOTS' numbered from 1 to 50 or you can push and pop images from a SLOT STACK. When restored to the screen, the image will appear before the user literally instantly.

Using the SLOTS feature, you can add a major enhancement to your application in minutes. This is, the ability to call any other application from anywhere within any application. By telling HostAccess to save the current screen image to a SLOT, your application can go off and execute other programs. Upon returning to your application, you simply request HostAccess to redisplay your screen image from that SLOT instantly.

SLOTS AiF Sequences

ESC [= N p Save the current screen image to slot number N.

ESC [= N q Restore slot N to the screen.

Where:

N Is a SLOT number from 1 to 50.

ESC [= p Push the current screen image on to slot stack.

ESC [= q Pop screen image from slot stack.

Note: the AiF Sequences **ESC [= 99 p** and **ESC [= 99 q** are used by the PUSH/POP environment

SLOTS Examples

At any input point the application should allow the user to invoke another application and then, on quitting that application, return to the original with the screen exactly as the user left it (including cursor position, colour attributes, etc.). Using AiF's SLOT feature, the Host application can issue the following AiF sequences:

```

    if input is 'run another application' then
    PUSH SLOT send to PC 'ESC[=p'
    run 'another application'
    return from other application
    POP SLOT send to PC 'ESC[=q'
    end if

```

This 'code' pushes the current screen image into the SLOT STACK and pops it back as soon as the user returns from the other application.

The screen image will be restored instantly, with all of the screen attributes, colours and cursor position set exactly as the user left them.

Using Push and Pop SLOTs applications can call other applications indefinitely with each application pushing and popping its own screen images without conflict with previous applications.

An application frequently makes use of a data entry screen. Rather than sending this screen to the PC each time it is required, the application could save it in a numbered SLOT at the start of the application.

```
screen_nbr = 1
send to PC data_entry_screen
send to PC 'ESC[=' screen_nbr 'p'
```

When the data entry screen is next required, the application only needs to send 'ESC[=' screen_nbr 'q' to instantly redisplay this same screen. The screen image will remain in the Slot for the entire HostAccess session, until overwritten by another screen or until the slots are cleared.

The SLOT STACK Facility

If there is a risk of applications clashes when using the numbered SLOTs, developers are recommended to make use of the SLOT STACK facility. As users move between applications (or areas within an application) and the application knows that the user will return to the previous screen, it is simpler to push and pop the required screen images from a SLOT STACK.

Each SLOT (screen image copy) requires approximately 4K of the PC's memory. There is no limit to the number of screens that may be pushed into a SLOT STACK other than the size of available memory in the PC. In practice, applications that push screens down more than six levels will tend to lose the user (user's memory's are limited too!).

Push SLOT will save the following screen related information together with a copy of the screen image:

- Cursor position.
- cursor status (shape, on/off).
- Screen attributes (colour, flashing, etc).
- Background fill character and attributes (as used for clears, clear screen, end of line, etc).
- Wrap and field modes.

Note: If your application needs to save more than the current screen and related information, see Save Environment on page 147.

FORMs

HostAccess enables you to store any host output on the PC's Floppy or Hard Disk. This output can be a whole screen, partial screen, AiF menus or selection boxes, or anything that you might output frequently. The output is stored in any specified form numbered from 1 to 255. Requesting HostAccess to display a form results in the output being processed at high speed. Whether the user is running at 1200 or 19200 baud, forms will always appear at the same fast speed.

Note: Characters with an ASCII value greater than 127 can be stored in forms, thereby accommodating special characters required for some languages, such as French.

FORMs AiF Sequences

Use the following sequence to write to a FORM file.

ESC [= Fn ; Fv ;1 s TEXT ESC [= s

Use the following sequence to process from a FORM file.

ESC [= Fn r

Where:

- Fn** Is the Form Number as an integer from 1 to 255.
- Fv** Is the Form Version as an integer from 1 to 255. The ';' is mandatory.
- s** Is the lowercase letter s - AiF code for FORMs.
- TEXT** Is the host output to be processed (and usually displayed on the screen). This may contain any valid screen display sequence (for example, cursor addressing, attribute setting) including other AiF sequences (such as FREEZE ON/OFF, load AiF Selection Box, etc.)

Note: This AiF sequence is terminated by the **ESC [= s** sequence and not the standard AiF terminator. This is because other AiF sequences can be stored within a form.

FORM files

The default DOS file where FORMs are saved to and restored from is called HOST.FRM in the directory in which HostAccess is running.

To specify an alternative FORM file, use the following sequence:

**ESC _ F DOS_form_file_name ESC **

This DOS_form_file_name may include a full DOS pathname.

To clear the currently active FORM file of FORMs use the following sequence:

ESC [= s

When a Form is created, it is given a version number of 1. Each time a Form is updated, this version number is incremented by 1.

To request the version number from a FORM file, use the following AiF sequence:

ESC [= Fv ; 1 r requests version number for FORM number Fn.

This will return to the host the following response:

<STX> <CR> Fn <CR>

Where:

<STX> Is a special Start of Text character (ASCII decimal value 002).

<CR> Is carriage return (ASCII decimal value 013).

Fn Is the FORM version number as an integer between 1 and 255. If a version number of 0 (zero) is returned, this indicates that the last specified Form Number does NOT exist.

FORMs Examples

Applications tend to build screen images by combining a number of variables into one and then displaying this consolidated variable. To save whole or part of any variable that is to be displayed, host applications simply need to insert this variable into an AiF sequence to save a FORM.

For example, to save and restore a three line portion of a screen display into and from the FORM numbered 22 in the FORM file HELPTEXT.FRM (in the HostAccess directory), use the following code.

Assign FORM file with:

```
send to PC 'ESC_F' : 'HELPTEXT.FRM' : 'ESC \ '
```

Build Screen Display item:

```
screen = time : date : screen.heading
screen = screen : column1,row1 'line 1'
screen = screen : column2,row2 'line 2'
```

Now save this to PC's disk with:

```
send to PC 'ESC[=22;1;1s' : screen : 'ESC[=s'
```


At any point within the application, the screen can be redisplayed from the PC's FORM file as follows:

```
send to PC 'ESC[=22r'
```

Notes on FORM files

Imagine storing your applications screen images in different files on the PC in different languages.

FORMs are actually held in a format that when restored, is replayed through HostAccess as if the characters were being sent from the host application to HostAccess. Because this FORM information is coming from the PC's disk, it is faster than having to send the same screen information from the host system. However, this does mean that FORMs do need to be compatible with the Terminal Type currently being emulated by HostAccess. In other words, if an application is running in VT100 emulation mode through HostAccess, any FORMs that are to be displayed must have previously been saved in VT100 emulation mode.

The number of FORM files is only limited by the available disk space on the PC. There is no limit to the number of FORM files that may be addressed by AiF FORM sequences.

It is important to realise that FORM files may contain any AiF sequence. This enables applications to store complete Selection boxes and/or Pop-Down Menu structures/selection lists on the PC's hard disk with obvious performance advantages when it comes to loading the menus. Diskless workstations would simply load such FORM files from the network fileserver's hard disk (say, from the same DOS directory from which HostAccess was invoked). In this environment, the host application must also manage the FORM files and their version numbers to ensure that they are present and that they contain the correct menu structures/selection lists. An AiF sequence is also provided to verify the existence of DOS files.

If you try to read a FORM that does not exist, HostAccess just ignores the request. If a FORM already exists, a write FORM sequence will overwrite it. The maximum size of a FORM is currently 32K, the maximum size of a form file is 64K. If you attempt to write a FORM greater than this size, it will be truncated, resulting in a corrupt screen when displayed.

If your screen is corrupted when you display a FORM, check that you had some form of flow control set when you loaded the form. If, for some reason, you are unable to use flow control, ensure that there is a delay in your program after sending each FORM to give HostAccess time to write the FORM to disk.

Freeze On/Off

This simple but effective feature available under HostAccess gives end users the appearance that their host machine's performance has been significantly improved.

It enables host applications to temporarily suppress screen output from HostAccess whilst building a new screen in the PC's memory and then to instantly release this screen output into view.

Freeze On/Off AiF Sequences

ESC [= 1 h Will FREEZE the screen.

ESC [= 1 I Will display data sent during FREEZE.

Currently, most applications display information on a data entry screen one field at a time. If the host system is slow, the user can actually see each field being displayed, in bursts on the screen.

Imagine being able to see all of this data appearing instantly on the screen. That is exactly what HostAccess's freeze on and freeze off facility provides. Before sending the first field, you turn freeze on, all subsequent data sent to the screen is not actually displayed.

After the last field has been output, you turn freeze off. This makes all the changes appear instantly on the screen. This technique can be used when clearing fields, printing boxes, drawing logos etc.

Notes on Freeze On/Off

When you send the freeze sequence to HostAccess, a flag is set telling HostAccess not to update the PC screen until the unfreeze code is received. Any data received from the host after the freeze command is processed, will update the screen image and back pages in memory but the physical screen is not changed. When the unfreeze command is received and the screen image in memory is used to update the real screen, this will happen instantly.

When used with HostAccess SLOTS, it can provide the application developer with the facilities of a multi-page terminal. The second page can be placed in a numbered HostAccess slot. It is updated by freezing the screen, pushing the current screen on to the SLOT STACK, pulling the second page on to the screen, updating it, putting back into its numbered SLOT, popping the original screen from the SLOT STACK and unfreezing the screen. Yes, that does sound like a lot of processing but HostAccess is so fast at moving screens between SLOTS and the number of characters that have to be sent to achieve the above listed functions is so small that this approach is very practical and very quick.

The use of freeze on/off is often governed by subjective assessments of how the application screens should be presented to users.

One simple means of implementing this feature is to adopt the philosophy that the application should turn freeze on immediately AFTER each user input and turn freeze off just BEFORE each user input. This tends to make all screen output appear as if it is all instant. But, this can also mean that on very slow host systems or when the user is accessing the system through a very slow link (say at 2400 baud), the application may leave the user with a 'frozen' (blank or unchanging) screen while HostAccess is waiting for the screen data and then the freeze off sequence to be sent down the line.

Many applications have catered for this type of 'speed problem' by displaying a standard message such as 'Now processing ... please wait' whenever the user selects an option and the screen subsequently needs to be changed. Of course, with HostAccess's AiF you could now show that message in a shadowed coloured box!

As an aid to developers implementing this freeze feature, we have added a special hot-key ALT/U which will immediately unfreeze the current screen image. This can be very useful when your program forgets to send the unfreeze sequence and you have spent a few seconds watching a blank PC screen and wondering just what your wonderful new release of software is doing!

Host Echo On/Off

It is useful on occasions to be able to suppress the host system's echoed output. An AiF sequence is available for this.

ESC [= 13 h Will enable host echo output to the screen.

ESC [= 13 I Will discard host echo output to the screen.

This will discard any text or cursor movement output from the host. It does not suppress host output to the system message line.

Examples

This AiF sequence may be used to replace the host ECHO ON/OFF, HUSH ON/OFF commands. Where these are not known (or are different for different host systems), this AiF sequence gives applications a consistent method of suppressing and enabling screen output.

Notes: Remember to enable host echo after suppressing it, if you want your users to see anything on the screen.

This sequence will not suppress any other AiF sequences output by the host, i.e. AiF sequences to open windows, update the system message line, etc. will still be carried out.

This is not the same as turning the host's echo off since characters are still echoed from the host to HostAccess.

Applications Enhancement

Any application is used and viewed by the user through the screens that the application displays. So applications should have 'functional attractiveness'.

Application screens should display screen information as concisely as possible and attract the user's attention to the correct part of the screen, as required.

The following sections show how to improve screens using Box and Line drawing.

Advantages to Developers

Because all of HostAccess's AiF features are actually processed within HostAccess on the PC, developers using these features will gain substantial benefits in the following areas:

- Reduced I/O burden on host system.
- Minimal host applications code required to achieve sophisticated screen displays (with consequent reductions in software maintenance).
- Fast implementation of the AiF features within existing and new host applications.
- Easy to code and easy to support other (dumb) terminals within the same application.

Box Drawing

Using AiF's Box Drawing feature, application screens can be very effectively enhanced by combining boxes and colour. AiF Box Drawing sequences are typically less than 18 bytes long - compare that with the number of characters conventionally required to draw boxes on host application screens! Boxes may be optionally framed, shadowed and/or exploded.

Use the following sequence to draw a box:

ESC [= Y1 ; X1 ; Y2 ; X2 ; BT ; A1 ; ... ; An x

Where:

- Y1** Is the top left-hand row.
- X1** Is the top left-hand column.
- Y2** Is the bottom right-hand row.
- X2** Is the bottom right-hand column.

Setting X1 and Y1 to 1 will display a box starting at the top left-hand corner of the screen.

If the X1 and Y1 parameters are set to 0, the box will be centred within the currently active window (or screen, if no window active). The box's dimensions are then determined by the absolute values of X2 and Y2.

BT	Describes the box type:
0	No frame.
1	Single line frame.
2	Double line frame.
3	Single line at top and bottom, double at sides.
4	Double line at top and bottom, single at sides.
64	Shadow window.
128	Explode window.

The last two values for BT are additive, e.g. a single line framed box that is exploded and shadowed has a BT value of 193 (1 64 128).

A1 - An Are optional parameters to set the colour of the box. If not present, the current attribute is used.

Box Drawing Application Examples

To draw an unframed exploding box of dimensions 3,3 to 10,40 with a background colour of Cyan and a foreground colour of Yellow, use the following sequence:

```
ESC [ = 3 ; 3 ; 10 ; 40 ; 128 ; 0 ; 1 ; 33 ; 46 x
```

Where:

3;3;10;40	Are the box co-ordinates.
128;	Is the box type of exploding unframed.
0;1;33;46	Are the parameters to assign colours (note the 1; is used to set the high intensity bit so that Yellow is generated by an attribute setting of 33, not Brown).

Notes: The frame is drawn round a box of the requested dimensions, i.e. area of the screen covered by the framed box is larger than the requested dimensions. The box is drawn in the requested colour or, if no colour is specified, the current screen colour attributes are used.

AiF windows are similar to boxes and should be used if the host application needs the ability to restrict further output to within the dimensions of the Window (box), without effecting the underlying screen. AiF box drawing always updates the underlying screen and will clear the area "under" the box. To add a frame effect around an area of the screen, without clearing this area, you can use either an AiF window (with the no clear on open option) or the AiF line drawing sequence (with the line co-ordinates set to those normally used for a box).

Line Drawing

Many terminal protocols support line drawing characters. However, just drawing a line across the screen involves sending 80 or so characters to the terminal. To draw a complex form would involve

several lines of code (especially if trying to cope with merging lines) and possibly thousands of characters being sent to the screen.

AiF's enhanced line drawing commands enable you to draw complex lines and frames with very simple commands that involve sending only a few characters from the host application program. Lines can be specified as single or double and HostAccess will intelligently merge new lines with any existing lines on the screen, if requested.

Use the following sequence to draw a line:

ESC [= Y1 ; X1 ; Y2 ; X2 ; LT ; A1 ; ... ; An z

Where:

- Y1** Is the top left-hand row.
- X1** Is the top left-hand column.
- Y2** Is the bottom right-hand row.
- X2** Is the bottom right-hand column.

Setting X1 and Y1 to 1 will draw a line starting at the top left-hand corner of the screen.

If either pair of X1,,X2 or Y1,Y2 parameters are set to 0, the line will be expanded to the full width of the currently active window (or screen, if no window active).

To draw *horizontal* lines make Y1 equal to Y2.

To draw *vertical* lines make X1 equal to X2.

LT Describes the line type:

- 0 Single line.
- 1 Double line.
- 2 Single line with merging.
- 3 Double line with merging.

The merging option requests AiF to intelligently merge the line or frame with any other line characters it meets or crosses, i.e. adding the appropriate 'T's, crosses, etc.

A1 - An Are optional parameters to set the colour of the line. If not present, the current attribute is used.

Line Drawing Application Examples

To draw a double line frame of dimensions 3,3 to 10,40 in Yellow on a Black background, use the following sequence:

ESC [= 3 ; 3 ; 10 ; 40 ; 1 ; 0 ; 33 ; 1 ; 40 z

Where:

- 3;3;10;40** Are the box co-ordinates.
- 1;** Is the draw double line option.
- 0;33;1;40** Are the parameters to set the colour attributes.

To draw a horizontal line from 5,3 to 5,40 (i.e. on row 5 from column 3 to 40), merge with any existing lines on the screen and use the current screen colours, use the following sequence:

ESC [= 5 ; 3 ; 5 ; 40 ; 2 z

To draw a horizontal line the full width of the screen (or currently open window) on row 11, use the following AiF sequence:

ESC [= 11 ; 0 ; 11 ; 0 z

Note: If the line co-ordinate parameter X1 is not equal to X2 and/or Y1 is not equal to Y2, a 'frame' will be drawn. There are certain advantages to using this method of drawing frames as opposed to using the AiF Box Drawing sequence. Firstly, the area embraced by a line drawn 'box' will be left intact (not cleared as in box drawing). Secondly, the borders of this line drawn box may be intelligently merged with other lines on the screen.

System Message Line (Line 25)

Most terminal emulations provide a system message line and HostAccess's emulations will support these. For those that do not support this feature, for example VT100, there is an AiF sequence to provide this facility.

ESC [= C1 ; A1 ; ... ; An w message CR

Where:

- C1** Is the starting column number.
- If C1 is absent or zero, the system message line is cleared and the column set to 1. Otherwise the line is left unchanged and the column set as specified. Characters are displayed until a carriage return is received or until the last column has been written to.
- A1 .. An** Are parameter settings to change the colour attributes on the System Message line. If not specified, then the current screen colours are used.

- Message** Is the text required on the System Message Line.
- CR** Is carriage return to terminate output to the System Message Line.

System Message Line Application Examples

To display the message MAIL WAITING in White text on a Cyan background at column 20 on the System Message Line, use the following sequence:

ESC [= 20 ; 0 ; 1 ; 37 ; 46 w Mail Waiting CR

The colour parameters are 0;1;37;46, and **w** is the AiF code for the System Message Line sequence.

To clear the System Message Line, use:

ESC [= w CR

Notes: While in System Message Line mode, message text can only consist of standard displayable characters. Non-displayable codes will terminate the System Message Line.

If the HostAccess status line is being displayed, it is switched off. The System Message Line is held in memory and redisplayed when the status line is switched off, i.e. the user may toggle between the status line and the System message line.

Application control of the system message line display is now available through an additional AiF sequence documented in the following section. This enables applications to restore the status line without the need for the user to redisplay the status line (via the Configure menu).

System Message Line Control

Full control of the System Message Line display can now be achieved by host applications using the AiF sequences below.

HostAccess's own status line can readily be restored as required by applications after they have used the System Message Line for their own messages (by using the AiF sequence described in the previous section).

- ESC [= 11 h** Forces the display of the current (application) System Message Line.
- ESC [= 11 l** Forces the display of the HostAccess Status Line, if this was enabled when HostAccess was loaded.

System Message Line Control Examples

An application will often use the System message line to display its own status information. On exit, it can now restore the user's HostAccess Status Line display (on the same line as the System Message Line).

```
display System Message Line
  send to PC 'ESC[=0w ..job UPDATE.BALANCES
  started at 12:22:15 ..'
on exit from the program, restore the HostAccess Status line
  send to PC 'ESC[=111'
```

Note: If HostAccess's Status Line Display has been disabled through HostAccess's configuration menus, it will not be possible to redisplay it with AiF sequence above.

Screen Modes, Including 132 Column Support

Host applications may switch the screen in to and out of these screen modes as required by using the AiF sequences below.

ESC [= 3 ; n h turns specified screen mode on.

Where:

n Is the screen mode value as defined below:

Mode	Rows x Cols	Monitor/Card
0	132 x 24	VGA cards only
1	80 x 24	(All cards)
2	80 x 42	(EGA cards only)
3	80 x 49	(VGA cards only)
5	132 x 25	(VGA cards only)
6	80 x 25	(VGA cards only)
7	80 x 43	(EGA cards only)
8	80 x 50	(VGA cards only)
9	40 x 24	(All cards)
10	40 x 25	(All cards)

Screen Modes Examples

A user wants to view a report before printing it. The host application can set the screen to 132 columns as follows:

```
set host terminal width to 25 rows by 132 columns
send to PC 'ESC[=3;5h'
    display and page through report
send to PC 'ESC[=3l'
set host terminal width back to 25 rows by 80 columns
```

Note 1 If mode parameter is null, it will have the same effect as setting mode to 1.

ESC [= 3 l returns screen mode to the settings as configured by the user.

Host applications should first ensure that the user's PC can support the required mode. Invalid modes will be ignored by HostAccess.

Note 2 HostAccess supports a variety of screen modes with the appropriate card and monitor. However, users should be aware that not all card and monitor configurations can support all of the modes shown above.

Note 3 The PC must be capable of supporting the desired screen modes. The VGA type will need to have been configured to the correct VGA BIOS type (within HostAccess's configuration menus) before trying to switch into 132 column screen mode.

Note 4 When you change screen modes, you reset a number of session parameters including closing any open AiF windows, AiF menus and clearing screen backpages, slots, etc. If your application needs to preserve the current environment before changing screen modes, use the Save Environment AiF sequence described on page 147.

Changing Cursor Shape

Host applications may change the cursor shape from a line in to a block and vice versa by using the AiF sequences below.

ESC [= 4 h Selects BLOCK cursor.

ESC [= 4 l Selects UNDERLINE cursor.

Changing Cursor Shape Examples

This simple sequence can be useful when writing routines that toggle between different input modes depending upon what the user is currently doing. For example, many DOS products will use a block cursor if the user is in Overwrite mode, or an underline cursor, if the user is in Insert mode. This sequence can be used to emulate this requirement within host applications.

Note: Some of HostAccess's terminal emulations (such as Wyse 60) already support this facility. However, the above AiF sequence makes this feature available in all of HostAccess's emulations.

Switching Cursor On/Off

Host applications may switch the cursor on or off as required by using the AiF sequences below.

ESC [= 10 h Switches cursor ON.

ESC [= 10 l Switches cursor OFF.

Cursor Application Examples

This simple sequence is useful when writing routines that need to hide the cursor for some reason or another. For example, when displaying error messages within a window it is nice to suppress the cursor and print a 'press any key' message in the window's footing.

Note Some of HostAccess's terminal emulations (such as Wyse 60) already support this facility. However, the above AiF sequence makes this feature available in all of HostAccess's emulations.

Changing the Screen Fill Character

The space character is normally used for clear screen, clear line and new window operations. This space character can now be replaced with any character in the standard IBM PC character set.

Host applications may change the fill character as required by using the AiF sequences below.

ESC [= 12 ; nnn h Sets the fill character.

Where

nnn Is the decimal value for the required IBM PC character.

ESC [= 12 l Resets the fill character to a space.

Screen Fill Character Examples

Visually attractive backgrounds to screens and windows can be created with this AiF sequence. For example, to fill a screen with musical notes as the background, use the following sequences:

Open window (use AiF window sequences and assign window colours)

```
send to PC 'ESC[=12;014h'  
send to PC clear screen code
```

Notes: The fill character ONLY applies to the currently open window (or the current screen if no windows are open).

In general, this facility should only be used for backgrounds. Very effective screens can be created by using an appropriate fill character over the whole screen and opening a Selection Box (or Input Box/Window) in the centre of the screen.

After filling a window, close it to turn the fill character off.

Using Alternate PC Fonts

Terminal emulations generally restrict the range of characters that can be displayed to a selected subset of the PC Fonts table. There are many occasions when applications need to be able to display other characters, such as currency, foreign language, scientific symbols and so on.

To display any character from the standard PC Fonts table, use the AiF sequences below.

ESC [= 9 ; n h Switches to the specified font table **n**.

Where:

	Table	Start	End	Offset
n	= 1	032	127	0
	= 2	160	255	-128
	= 3	000	031	32
	= 3	128	159	-64

The character values above are for decimal ranges.

To display any character from within a specified range, the application should switch on the appropriate font table and display the character for the required character value plus/minus the offset.

ESC [= 9 I Resets the table back to the terminal font.

PC Font Examples

When presenting choices of selections from a pop-down menu or selection box, it is helpful to be able to display the PC's up/down arrow key symbols to indicate which keys the user should use.

To do this use the following sequences:

```
Up_Arrow_display = character value of 2432
Down_Arrow_display = character value of 2532

send to PC 'ESC[=9;3h'
send to PC Up_Arrow_display at required cursor position
send to PC Down_Arrow_display (next to up arrow)
send to PC 'ESC[=9I'
send to PC " keys to select menu item"
```

Note: Emulation specific details such as cursor positions are handled separately from characters to be displayed on the screen. In general, it is better to switch in to the required font, display the required characters at defined screen positions and then switch back to the normal terminal font. In other words, there is no need to switch in to and out of the font for each special character.

Special Output Mode

There are occasions when output to the screen will attempt to address areas of the screen that are outside of the currently open window.

Host applications may now suppress output to these areas of the screen by using the AiF sequences below.

ESC [= 5 h Suppresses any screen output addressing areas outside the currently open window. Output will be continued when the cursor is repositioned to a valid co-ordinate (i.e. within the current window).

ESC [= 5 l Disables this special output mode.

Special Output Mode Examples

This AiF feature can be useful when 'GUIising' host applications over which one has limited control. For example, it might be possible to add modules into such an application and make these modules more presentable by using some of HostAccess's AiF features such as windows, boxes and so on. However, it is quite possible that core routines within the original application will still insist on addressing areas of the screen outside the new module's windows, e.g. to display system/error messages. In these instances it is useful to be able to suppress this screen output which would otherwise tend to corrupt the display within the new module's AiF windows.

Centering Text

Any text string can be centred on a given line within the currently open window (or screen, if no windows are open) without the need for the host application to work out the starting cursor column address.

Host applications can center text by using the AiF sequence below.

**ESC _ Y1 C text ESC ** Centres text within the current window (or screen).

Where:

- Y1** Is the row (line) number within the currently open window (or screen) on which the text should be centred.
- C** Is the literal capital 'C': AiF code for centre.
- text** Is the text string to be centred.

Centering text example

Help text for an application might consist of, say, 7 lines held within an array. To display the help text centred within a window, use the following AiF sequences:

Assign help text array (read from file, etc.):

```
help_text(1) = 'help line 1'
help_text(2) = 'help line 2'
...
help_text(7) = 'help line 7'
```

Open the help text window using colours yellow on red:

```
send to PC 'ESC[=10;10;17;70;193;0;1;33;41w'
```

Loop through the text displaying it:

```
row = 1
loop
until row greater than 7
send to PC 'ESC_' row 'C' help_text(counter) 'ESC \'
row = row + 1
repeat
```

Wait for user acknowledgement and then close the window (with a window clear option)

Notes: Obviously, text strings wider than the current window (or screen) cannot be centred, but are truncated to fit within the window.

Using Macros

HostAccess's macro language enables you to open windows and display messages even before your users are connected to their host applications. Because macros themselves support AiF sequences, it is easy to build macros with AiF features such as colour, boxes, and windows.

Any macro may be invoked from a host routine by using the following AiF sequence:

**ESC_ s macrotext ESC **

Where:

s AiF delimiter, as a lowercase 's'.
macrotext Is the text of the required macro, with a carriage return char (13) separating each line.

See Chapter 5 - Using the Macro Language for details on writing macros.

Keyboard Control Features

You can control loading and modifying the Function Keys available on any standard IBM or compatible PC Keyboard.

Most applications will normally relieve the user of having to manually program application specific Function Keys.

HostAccess gives host applications up to 48 programmable Function Keys. Each Function Key may be individually loaded with any ASCII character sequence, including control characters.

Note: HostAccess supports international keyboard mapping for the USA and all European countries.

Programmable Function Keys

Any one or all of the forty Function Keys available in HostAccess may be programmed by a host application to send character sequences to the host as if they were entered from the keyboard. These character sequences may consist of any ASCII character including control codes.

Programmable Function Key Table

Keyboard Keys	AiF Programmable Key Number				Keyboard Type
	Normal	Shifted	Ctrl	Alt	
Function Keys					
F1 to F10	1 - 10	11 - 20	21 - 30	31 - 40	All
F11	41	43	45	47	AT
F12	42	44	46	48	AT
Arrow Keys					
Up arrow	49	53	57	61	All
Down arrow	50	54	58	62	All
Left arrow	51	55	59	63	All
Right arrow	52	56	60	64	All
Edit Keys					
Insert	65	71	77	83	All
Delete	66	72	78	84	All
Home	67	73	79	85	all
End	68	74	80	86	all
Page Up	69	75	81	87	all
Page Down	70	76	82	88	all

Programmable Function Key AiF Sequence

Use the following AiF sequence to program a function key.

**ESC_n K Key data ESC **

Where:

- n** Is the programmable Function Key Number. If this is set to 0 then all programmable keys will be reset.
- Key data** Is the character(s) required to be sent to the host when the user presses the Function. If this string is empty the key **n** will be reset.

Key. Key data should be entered as normal text (without quotes). Control characters are entered as ^A, ^B, etc. Use ^^ for the character '^'.

For characters in the range 128 to 255 enter the three digit decimal value after a '^' e.g. ^128. (Any character may be entered in this manner, but please note that 7 bit links will not send 8 bit characters - the top bit will be stripped off).

Examples

To program Function Key 1 to send the word SYSPROG, then a Carriage Return, then the word MENU followed by another Carriage Return, use this sequence:

ESC_1KSYSPROG^MMENU^MESC

or:

ESC_1KSYSPROG^013MENU^013ESC

Programming Control Codes

Notes You can program any character into the function keys by using a top-arrow ^ followed by a 3 digit number. BACKSPACE for example would be ^008, character 254 would be ^254.

Some useful control codes are listed below:

Programmable Sequence	Description	Keyboard Input
^I	control I	Tab
^J	control J	line feed
^M	control M	Carriage return
^[control [ESCape

Note that some XT compatible machines may not generate a code for the function keys F11 and F12, even though these keys may be on the keyboard.

You should be aware of the order of precedence assigned to Function Keys, depending upon how they have been loaded.

Toggling Caps Lock On/Off

Host applications may switch the caps lock on or off as required by using the following AiF sequences.

Toggle Caps Lock AiF Sequence

ESC [= 28 h Toggles Caps Lock on
ESC [= 28 l Toggles Caps Lock off

Switching Scancode Keys On/Off

The PC keyboard has a number of keys that are generally accessible to the user when using DOS products but are not generally accessible when using host applications. This is simply because there may not be any matching keys in the terminal being emulated. Virtually all special keys, such as arrow keys, Page Up/Down, Ins, Del, Ctrl, Alt, etc., are now accessible to host applications by using the PC Scancode keys facility with the following AiF sequences.

This AiF sequence is available to host applications regardless of what terminal emulation the user has chosen.

Scancode AiF Sequence

ESC [= 6 ; p h Switches Scancode Keys on.
ESC [= 6 l Switches Scancode Keys off and keyboard inputs revert to the characters that would normally be returned by the user's current emulation.

p is the scancode prefix as the ASCII decimal value of the character to precede the keyboard response. The default value of **p** is zero, but we recommend that **p** is set to **2** (ASCII character STX, start of text) so that this is consistent with all of the other AiF sequences that send responses.

The prefix is needed so that applications can easily determine that they should be looking for Scancode keys when processing user input. When switched on, the user keyboard responses are sent back to the host in the following format:

scancode_prefix key_scancode

Where:

Scancode_prefix Is the ASCII decimal value of character used as prefix.
Key_scancode Is the ASCII character corresponding to the key depressed by the user. A list of Scancodes supported is shown below.

Note: some UNIX systems are unable to accept ASCII character value 0 as valid input.

Scancode Keys

Ranges of keys are specified as from the leftmost key to the rightmost key on one row of the keyboard. For example, Alt/Q to A/Q to Alt/P is the range of keys generated when the Alt key is pressed at the same time as one of the following Q,W,E,R,T,Y,U,I,O,P keys.

HostAccess supports the following Scancode keys. Any other key(s) entered by the user whilst Scancode keys are on are returned in their normal character representation.

Keytop Legend (keystrokes)	Scancode in Hex	ASCII Character decimal value
Alt Esc	01	1
Alt Backspace	1E	14
Shift + Tab	0F	15
Alt/Q to Alt/P	10 to 19	16 to 25
Alt [1A	26
Alt]	1B	27
Alt Enter	1C	28
Ctrl	1D	29
Alt/A to Alt/L	1E to 26	30 to 38
Alt/Z to Alt/M	2C to 32	44 to 50
Alt	38	56
Function keys 1-10	3B to 44	59 to 68
Home	47	71
Cursor Up	48	72
Page Up	49	73
Alt Num - (minus)	4A	74
Cursor Left	4B	75
Cursor Right	4D	77
Alt Num + (plus)	4F	78
End	4F	79
Cursor Down	50	80
Page Down	51	81
Ins	52	82
Del	53	83
Shift Function keys 1-10	54 to 5D	84 to 93
Ctrl Function keys 1-10	5E to 67	94 to 103
Alt Function keys 1-10	68 to 71	104 to 113
Ctrl/Print Screen	72	114
Ctrl/Cursor Left	73	115
Ctrl/Cursor Right	74	116
Ctrl/End	75	117
Ctrl/Page Down	76	118
Ctrl/Home	77	119
Alt/1 to Alt/+	78 to 83	120 to 131
Ctrl/Page Up	84	132
F11	85	133
F12	86	134
Shift F11	87	135

Keypop Legend (keystrokes)	Scancode in Hex	ASCII Character decimal value
Shift F12	88	136
Ctrl F11	89	137
Ctrl F12	8A	138
Alt F11	8B	139
Alt F12	8C	140
Ctrl Up Arrow	8D	141
Ctrl Num - (minus)	8E	142
Ctrl Num 5	8F	143
Ctrl Num + (plus)	90	144
Ctrl Down Arrow	91	145
Ctrl Ins	92	146
Ctrl Del	93	147
Ctrl Tab	94	148
Ctrl Num /	95	149
Ctrl Num *	96	150
Alt Home	97	151
Alt Up Arrow	98	152
Alt Page Up	99	153
Alt Left Arrow	9B	155
Alt Right Arrow	9D	157
Alt End	9F	159
Alt Down Arrow	A0	160
Alt Page Down	A1	161
Alt Ins	A2	162
Alt Del	A3	163
Alt Num /	A4	164
Alt Tab	A5	165
Alt Num Enter	A6	166

Scancode Keys Examples

To determine if the user has depressed the Function Key F1, regardless of the current emulation being used and regardless of the possible contents of this function key (which may have been loaded by this or another application), use the following logical construct:

```

switch scancode keys on
  send to PC 'ESC[=6;2h'
get_user_input
  input user_response
  if first character of user_response equals ASCII 002 (decimal)
  then
    scancode_key = 2nd character of user_response
  end if
find out which key has been pressed

```

```
    if scancode_key equals ASCII 059 (decimal) then
        F1_pressed = true
    end if
process keyboard responses
    if F1_pressed is true then
        send to PC "you pressed the F1 key..."
        and so on ....
```

Do not forget to switch Scancode keys off just before exiting this routine, with the following sequence:

```
send to PC 'ESC[=61'
```

Scancode Keys Notes

It is important to remember to switch Scancode keys OFF when your application exits. If they are not switched off, other applications may not be able to interpret user input correctly.

You should not switch Scancode keys on and off around individual input statements as this cannot be done fast enough for typeahead. In general, switch Scancode keys on when entering a routine and switch off when exiting.

The Scancode prefix has been made a parameter so that you can change this to suit the host system or network on which their applications are being used.

You can use this with the Page Keys facility of HostAccess to give more flexible keyboard re-mapping and input.

If Scancode keys mode is on, the scan codes are sent to the host in preference to any other value associated with that key. Where Function keys are concerned, Scancodes take precedence over host or user programmed function keys, etc.

Typeahead Mode

Users tend to like to be able to typeahead when running host applications, particularly impatient users or users that are familiar with the keystrokes required to get to a defined point within an application. However, when host applications start to use the AiF menus, selection boxes and field input modes, the user's typeahead keystrokes may be sent to the host (rather than to HostAccess) between successive AiF sequences.

To prevent this happening, host applications can switch on a special input mode that enables the user's typeahead keystrokes to be saved for and used by HostAccess's menus and input modes.

Host applications can switch this mode on by using the AiF sequence that follows.

- ESC [= 20 h** Switches Typeahead Mode on.
- ESC [= 20 I** Switches Typeahead Mode off. Any characters within HostAccess's typeahead buffer will immediately be sent to the host.

Typeahead Mode Examples

The user may know that he/she is about to enter an application that makes use of HostAccess AiF menus and that two right arrows, then a down arrow, a carriage return and, finally, a Function Key F5 will select the required menu option and accept the contents of an AiF input box.

By switching Typeahead Mode on, the host application will enable HostAccess's AiF to locally process all of the user's keystrokes, rather than send them to the host.

Notes

It is important to remember to switch Typeahead Mode OFF. If it is not switched off, other applications may not be able to interpret user input correctly.

There is a limit of 20 bytes in HostAccess's Typeahead buffer. In practice this is adequate as processing will almost invariably catch up with the user before the user has been able to type in 20 characters.

If the user presses the break key, HostAccess automatically switches Typeahead Mode off and flushes the PC's typeahead buffer.

Command Stack Control

HostAccess automatically records user keyboard input into a Command Stack within the PC's memory. The user is able to recall this Command Stack, modify entries within it (by using the ALT/R hot-key) and re-use previously entered (or modified) commands.

Host applications, in general, will not want to fill up the user's command stack with input required by their applications (such as data entry screens, etc.).

This AiF feature allows a host application to stop keyboard inputs being appended into the Command Stack and to re-enable this feature (normally on exit from the application).

Host applications can switch this mode on and off by using the AiF sequences below.

- ESC [= 21 h Enables the Command Stack.
- ESC [= 21 l Stops HostAccess putting keystrokes into the Command Stack.
- ESC [= 21 e Flushes the command stack

Command Stack Examples

The Command Stack has a limited number of entries, so in general, any application that requires keyboard input should disable the Command Stack in order to preserve the user's commands entered before invoking the application.

Note: Applications that process user input on a single character basis will tend to add entries into the command stack until the user enters a carriage return. Such applications should consider switching the command stack mode off while processing user input, then switch it back on when the user exits the application or temporarily leaves it through an application's gateway.

Mouse Control

You can program certain host applications so that users can use a pointing device, such as a mouse, to interact with host based software.

You can give host applications the ability to monitor mouse movements and button depressions by the user with the following AiF sequences.

You can also program Hotspots. A Hotspot is a character string on an emulation screen which has been programmed so that when you move the mouse cursor over the character string and click the right button, a particular function is activated. If the character string starts with the text "F1" through to "F12", the programmed value stored in that function key is returned to the host. If that character string is not detected, then the first alphanumeric pattern from the left hand edge is returned with a postfix of Carriage return, so that in the sequence A12B34, the string returned is "12^".

Mouse Control AiF Sequences

To detect if a mouse is installed on the user's PC use this AiF sequence:

ESC [= 8 n

The following response will be sent to the host application:

<STX> <CR> code <CR>

Where:

<STX> Is the special Start of Text character (ASCII decimal value 002).

Code Is the mouse install status where:

0 mouse NOT installed.

1 mouse installed.

<CR> Is carriage return (ASCII decimal value 013).

To activate the mouse or Hotspots and determine which events should be monitored, use the following AiF sequence:

ESC [= 27 ; n h Switches mouse monitoring or Hotspots ON.

Where:

n Is an integer code that determines which mouse events will be returned to the host, where:

1 Monitor Left Button pressed down.

2 Monitor Right Button pressed down.

4 Monitor Centre Button pressed down.

- 8 Continuously send mouse addresses whilst a button is depressed.
- 16 Switches on Hotspots.
- 65 Monitor Left button double click
- 66 Monitor Right button double click
- 68 Monitor Centre button double click
- 128 Monitor Scroll Wheel

Note: Where 16 is used, only 1,2 and 4 will work. If 16 is applied on its own, the mouse will not work as no button has been supplied.

Either mouse monitoring or hotspots may be enabled but not both. If the value of 16 is seen then Hotspots will be chosen in preference.

These codes are additive, e.g. to monitor the mouse continuously while the Left button is depressed set this code to 9.

Format of Events Returned

Mouse events are returned to the host application in the following format:

<STX> MS <CR> button_status , Y1 , X1 <CR>

Where:

- <STX>** Is the special Start of Text character (ASCII decimal value 002).
- MS** Is the literal letters 'MS' (AiF code for mouse).
- <CR>** Is carriage return (ASCII decimal value 013).
- Button_status** The state of the mouse for this monitored event code in the form an integer, Where:
 - 0 Only returned if "continuously" monitoring the mouse and then when the button(s) is/are released.
 - 1 Left button depressed.
 - 2 Right button depressed.
 - 3 Both Left and Right buttons depressed.
 - 4 Centre button depressed.
 - 5 Left and Centre buttons depressed.
 - 6 Right and Centre buttons depressed.
 - 7 Left, Centre and Right buttons depressed.
 - 65 Left button double clicked
 - 66 Right button double clicked
 - 68 Centre button double clicked
 - 128 The Mouse Wheel has been rolled. In this case, Y1 is the number of detents by which the wheel turned (a positive number indicates that

the data on screen should appear to move downwards) and X1 is the number of lines (as configured in Windows) that is suggested to roll per wheel detent.

Y1 The Y co-ordinate as an integer row value.

X1 The X co-ordinate as an integer column value.

Mouse monitoring and Hotspots should be turned off with the following sequence:

ESC [= 27 I Switches mouse monitoring or Hotspots OFF.

Mouse Interaction Examples

A host based calculator program could be dramatically enhanced by making use of AiF sequences, including mouse interaction. The basic structure of such a program is outlined below:

```

Send to PC AiF sequences to "draw" the calculator (boxes, symbols,
etc).
Build map of valid mouse co-ordinates.
Activate mouse and monitor any button with:
send to PC ESC '[=27;7h'
Loop to process user input
input calc_key
if calc_key starts with STX then
input mouse_coordinates
    map mouse co-ordinates into valid calc_key
end if
assign calc_operand from calc_key
process calc_operand until quit

```

Please note that in the above example that no check has been made to see if a mouse is installed and the routine attempts to handle both mouse and keyboard input at the same time by looking for the special Start of Text character. If this is found, mouse input is assumed to have occurred. If not found, keyboard input is assumed. Not all applications will need (or wish) to monitor inputs from both devices at the same time.

Notes: You should be aware that if you monitor mouse events continuously you may "flood" the host system with data from the mouse, with a detrimental impact on performance and user patience.

It is recommended that applications, particularly those running over asynchronous links or X25 networks, selectively monitor mouse events, e.g. only when the user depresses the left button.

Mouse interaction will automatically work with existing AiF Selection Boxes and Pop-Down Menus. If a user chooses to use the mouse whilst within an AiF menu, the mouse co-ordinates on the selected menu option are converted into the appropriate menu co-ordinates. There is no need to include the above mouse processing AiF sequences within existing AiF menu processing routines.

Programmable DOS Gateway

The AiF DOS gateway gives host applications the ability to invoke DOS and to run DOS applications. Upon Exiting from DOS, the user is returned to the host environment exactly where it was left, with the current screen, backpages, including the AiF menus, etc. intact.

This DOS interface is so crisp that it is possible to seamlessly combine the DOS and host operating environments.

Programmable DOS Gateway AiF Sequence

To invoke the AiF DOS gateway from a host application use the following sequence:

**ESC _ sc ; 0 D Cmd1 ; ... ; Cmdn % keys ; Cmdnn ESC **

Where:

- sc** Is the screen control code in HostAccess, as follows:
 - 0** Opens, activates and normalises a DOS shell window and executes the DOS routines within that window.
 - 1** Opens, activates and minimises a DOS shell window.
 - 2** Leave currently active host session screen.

When the DOS commands have been completed, the DOS shell window will be closed and the HostAccess window will be automatically reactivated.

- 0** Is the literal 0.
- D** Is the literal capital letter D.
- Cmd1 ..** If no commands are specified the user is taken to the DOS command line.
- Cmdnn** Entering EXIT will return the user to the host session. If specified, these may be any valid DOS commands. Any number of DOS commands may be strung together using the semi-colon as each command delimiter.
- Cmdn % keys** If DOS keyboard inputs are required to drive DOS applications then a special symbol '%' may be appended to a DOS 'command'. This turns on the DOS Keyboard Stacker and feeds the DOS applications with the keystrokes it requires. See DOS Keyboard Stacker on page 184 for details.

DOS commands sent by the application from the host should be in the same format as you would enter them at the DOS command line.

They can consist of DOS operating system commands (such as DIR or CD), program calls (such as WS for WordStar) or batch file calls.

DOS Gateway Examples

To change directory to your word-processing directory WP and then run your word-processor, use the following sequence:

```
ESC _ 0;9D CD\WP;WP ESC \
```

As soon as the last DOS command is finished, the host application screen will be returned to. This may mean that the user will not have an opportunity to read the output of commands such as DIR. However, DOS has a command called PAUSE which waits until the user hits a key. You can add this command to your command string to allow the user to read the screen before it is overwritten, for example:

```
ESC _ 2;0D DIR;PAUSE ESC \
```

This lists the contents of the current DOS directory and then waits for the user to hit a key after the 'strike a key when ready . . .' prompt.

DOS Keyboard Stacker

DOS Keyboard Stacker is a facility within the DOS interface of HostAccess which automatically places keystrokes into your PC's keyboard buffer and sends them to a DOS application as if they were being typed in by the user.

Almost any keyboard input may be simulated and delays can be included to overcome problems caused by DOS applications flushing the keyboard buffer before accepting input. This facility can be used with the AiF sequences described in the two preceding sections for programmable DOS gateways and running DOS programs.

Keyboard Stacker Description

Ordinary alphanumeric data, including numbers, punctuation, braces, etc., are stacked by placing them within single or double quotes on the command line as below:

% "document name"

Keys that do not correspond to a displayable character, for example control keys, are represented by special two character codes.

Special Keys: Mnemonics

A number of mnemonics are defined to represent certain special keys. These are:

Mnemonic	Special Key
LA	Left Arrow
RA	Right Arrow
UA	Up Arrow
DA	Down Arrow
PU	Page Up
PD	Page Down
HM	Home
EN	End
IN	Insert
DE	Delete
TA or TB	Tab
ST or BT	Shift Tab (=Back Tab)
ES	Escape
BS	Backspace

Mnemonic	Special Key
SP	Space bar
CR	Enter
LF	Ctrl-Enter
DQ	The double quote "
SQ	The single quote '

These codes can be entered in upper or lower case. You may use spaces between mnemonics to increase readability. These spaces will be ignored (unless they are between quotes).

Special Keys: Leader Characters

A number of keys (such as Shift, or function keys) can be represented by a special **leader character**, followed by a single character qualifier. For example, function keys F1, F2, F3 ...F9 are represented by F1, F2, F3 ...F9. Function keys F10, F11 and F12 are represented by F0, FA and FB respectively.

There will be times when a DOS application program or command will flush the keyboard buffer before asking for a keystroke. This is to force you to respond or to make sure the response is not accidental. If you just stack the keys you want, they will also be flushed out. An example of this is the DOS LABEL command.

You can place a delay into the stack, so the DOS Keyboard Stacker will pause for a specified period before continuing to insert keys into the buffer, using the command **Wnn**. For example, to wait about 2 seconds before putting an ESCape key into the buffer, use:

% W36 ES

Alternatively, to program a wait of about one minute, followed by an ESCape key, use

% W255 W255 W255 W255 ES

It is also possible to insert "pauses" within the keyboard stacker sequences, using the **WP**, **WE** and **WB** mnemonics, and these wait for user input before activating any subsequent stacked keys.

All these mnemonics are summarised below:

Character	Represents	Character	Represents
^	Control function.	Wnn	Wait time in 55 millisecond units (clock ticks, about 18.2 per second), where nn is from 1 to 255.
@	Alt function. Shift function.	WP	Wait for user key and then pass it on.
		WE	Wait for user key and then throw it away.
F	Function key	WB	Wait until key buffer is empty.
S	Shift function key	BR	Break.
C	Control Function key.		
A	Alt Function key		

For example, **^C** represents Control C, **@2** represents Alt/2, and **A9** represents Alt/F9.

Special Keys Example

Here is a simple example of a DOS command using DOS Keyboard Stacker. To execute the DOS TIME command, wait 1 second and then input a time of 12:10 followed by a carriage return, use:

```
ESC _ D TIME % W018 "12:10" CR ESC \
```

Printing to a DOS File or Device

All forms of terminal printing including screen dump, hardcopy and direct (slave) printing are supported.

Printer output can be generated via an AiF sequence from the Host or from within HostAccess, and can be directed to a DOS disk file or to a printer on the PC. The print destination can either be set through the Print Setup... option on the Session menu or from the host application using an AiF sequence. This destination name will only affect the currently active session.

All terminal emulation protocol specific printing commands are supported, for example McDonnell Douglas' PORTOUT. However, it is recommended that you use the ANSI sequences given below as they are supported in all the terminal emulations available (and so applications will only need to support one set of terminal printing sequences).

Printing AiF Sequences

- ESC [= 0 i** Print screen to current print device.
- ESC [= 4 i** Switch OFF direct (slave) printing.
- ESC [= 5 i** Switch ON direct (slave) printing to current print device.
- ESC [= 8 i** Closes the printer, even when the keep printer open feature is enabled.

To change the current DOS device for printing for the currently active session, use the following sequence:

**ESC _ L device.name ESC **

Where:

- Device.name** Is the DOS device or filename into which all print output should be directed for this session. It could be LPT1, LPT2, COM1 or COM2 if you have a printer on one of those devices or it could be a DOS file name. To reset to the default printer use PrintManager.

Printing Examples

To send print data to a printer on parallel port 1 use the following 'code':

```
SET DEVICE send to PC 'ESC_LLPT1ESC \'
PRINT ON send to PC 'ESC[=5i'
      send to PC print_data_lines
PRINT OFF send to PC 'ESC[=4i'
```

To switch printing to the DOS file C:\PRINT.LST and to append a dump of the screen contents to it, use the following 'code':

```
SET DEVICE send to PC 'ESC_LC:\PRINT.LSTESC \'
DUMP SCREEN send to PC 'ESC[=0i'
```

Notes: Although both ANSI and terminal protocol specific printing commands will be accepted, direct print On and Off commands should be matched. For example, you cannot use the PRISM specific command to switch on printing and the ANSI command to switch it off.

If you are using the Host Printing facility of HostAccess's File Services, do not attempt to direct HostAccess DOS printing to this device.

When printing to a DOS file, this file is *always* appended to. If this is not required, delete the file before printing commences. This may be done by using the Erase DOS File AiF sequence (for more information see the following section).

AiF has an additional 'printing' feature that enables screens to be sent back to the host system (for details see Capturing Screen Text on page 201).

Printing to a USB Printer

If you wish to direct print output to a USB printer you will need to instruct Windows to map the USB printer to a given LPT port.

You will first need to share the USB printer and then run the following command:

```
net use LPT1: \\ServerName\PrinterName /persistent:yes
```

Where:

ServerName Is the name of the server sharing the printer

PrinterName Is the share name of the printer

Erase DOS File

Host applications that pipe data to DOS files will often need to delete the target DOS file at some stage within their processing. Applications will want this to happen transparently to the user (i.e. without the need of invoking the DOS gateway and executing the DOS Del command).

This is easily achieved by using the AiF sequence that follows.

```
ESC _ E filename ESC \
```

Where:

filename Is the name of the DOS file to be deleted, including its filename extension and the full DOS drive:\path to the file.

Examples

To delete the DOS file called TEMP.DOC in the DOS drive:\path C:\HOST, use the following sequence:

```
send to PC 'ESC_E C:\HOST\TEMP.DOC ESC \'
```

Note: Use with care!

No indication or request for confirmation of the file deletion is given to either the user or the host application. It is the application's responsibility to verify that the correct file has been deleted, if this is required. If the DOS file to be deleted does not exist, control is simply returned uninterrupted to the host application.

See Verify DOS File or Directory Exists on page 192 if you need an AiF sequence to verify the existence of a DOS file.

Request HostAccess DOS Run Directory

Host applications can request that HostAccess tells them from which DOS directory HostAccess is running.

It is often useful for host applications to be aware of the current run-time DOS path so that this can be used to store temporary DOS files, for example, when automating file transfer.

This can be achieved by using the following AiF sequence:

ESC [= 9 {; code} n

Where:

Code	0* = Working directory, long file name format
	1 = Installation directory, long file name format
	2 = Temp directory, long file name format
	8 = Working directory, short file name format
	9 = Installation directory, short file name format
	10 = Temp directory, short file name format.

Note: Long file name support is available to both the 16 and 32-bit versions of HostAccess, but only on Windows 95/98 and NT.

HostAccess responds with the following message:

<STX> <CR> path <CR>

Where:

<STX>	Is a special start of text character (ASCII decimal value 002).
path	Is the full DOS drive and path to the HostAccess run-time directory, e.g. C:\HOSTACC\HOST.EXE
<CR>	Is a carriage return (ASCII decimal value 013).

Examples

To request the current HostAccess run-time directory, use the following sequence:

```
send to PC 'ESC[=9n'
loop input response until response equals STX do repeat
input pc_path
DOS_run_drive = first two characters of pc_path
DOS_run_directory = all characters after the last "\"
delimiter
in the pc_path string
```

Note: Remember that the current DOS session drive and path may be changed in a number of ways, by other AiF sequences, or by the PC user. This sequence is useful for host applications that need a consistently valid

DOS path for operations such as file transfer to DOS. Refer to the following section if you need an AiF sequence to verify the existence of a DOS file.

Request HostAccess System Information

This AiF sequence may be used to find out more information about the system running HostAccess.

Use the following sequence to get the HostAccess System Information:

**ESC _ 84 ; type w {variable} ESC **

Where:

type The type of information HostAccess should return:

1. Printer Information
2. Date and Time
3. Date
4. Time
5. Computer Name and User Name
6. Computer Name
7. User Name
8. Environment Variable

variable The environment variable. Used only when type = 8.

This returns:

<STX> <CR> data <CR>

Where:

<STX> Is the start of text character (ASCII decimal value 002).

<CR> Is carriage return (ASCII decimal value 013).

data Is the data returned e.g. Computer Name, User Name, Date and/or Time etc

Verify DOS File or Directory Exists

Host applications that manipulate DOS files (e.g. through HostAccess's file transfer, FORMs or direct print facilities) often need to check the existence of the target DOS file at some stage within their processing. Applications will want this to happen transparently to the user.

This is easily achieved by using the AiF sequence that follows.

**ESC _ G path ESC **

Where **G** is the capital letter 'G'.

path Is the name of the DOS file or directory to be verified, including its filename extension and the full DOS drive:\path to the file.

HostAccess answers this inquiry with a response in the following format:

<STX> <CR> code <CR>

Where:

<STX> Is the start of text character (ASCII decimal value 002).

<CR> Is carriage return (ASCII decimal value 013).

Code Is an integer code that answers the verification request as one of the following: 0 (DOS path does not exist), 1 (DOS File exists) or 2 (DOS Directory exists).

Verify DOS File Examples

To verify the DOS file called TEMP.DOC in the DOS drive:\path C:\HOST, use the following sequence:

```
send to PC 'ESC_G C:\HOST\TEMP.DOC ESC \'
loop until input_string equals STX do repeat
input response
if response equals 0 print "File does NOT exist !!"
if response equals 1 print "File exists"
if response equals 2 print "TEMP.DOC is a DIRECTORY !!"
```

Note: HostAccess verifies the existence of either a file or a directory dependent upon the DOS path specified.

Data Extraction to DOS and Windows

We have developed for PICK users a sophisticated and powerful tool for automatically sending PICK data to almost all DOS and Windows packages such as WORD, EXCEL, LOTUS, WordPerfect, SuperCalc, QUATTRO etc.

These routines now give developers the ability to integrate data seamlessly from PICK databases into DOS and Windows packages.

Consequently, there are many users today who are able to use PICK's query language to retrieve data and then pass this data directly into a DOS or Windows spreadsheet, automatically adjusting column widths and headings as the user is taken into the spreadsheet - and all of this by simply selecting a menu option on the PICK host!

UNIX users are provided with the file transfer facilities to enable them to extract and retrieve data to/from DOS files.

The following topics describe the AiF sequences used for communication with other Windows applications, and for monitoring their status. For details of AiF sequences to use within Dynamic Data Exchange, see Chapter 4 - Dynamic Data Exchange.

Displaying Images

HostAccess uses a separate Windows program to display images. Images are displayed within their own window which can be moved, re-sized, maximised, or minimised by the user, as required.

The Display Images program can be invoked using the Start Windows Program sequence (described later in this section), using the following command line parameters.

Displaying Images AiF Sequence

Use the following sequence to display an image:

IMAGE /I filename {/T title} {/Z zoom} {/F}

Where

- IMAGE** Is the name of the image display program (IMAGE.EXE) which will have been installed into the directory in which HostAccess was installed.
- /I** Is the command line flag indicating that an image filename will be specified. This flag *must* be followed by a space.
- filename** Is the full path and filename for the .PCX image file. There is no need to specify the .PCX suffix.
- /T** Is the optional command line flag indicating that a title will be specified. This flag *must* be followed by a space.
- title** Is an optional title that will be displayed in the Application Name bar. If omitted, "Image [Filename]" will be used.
- /Z** Is the optional command line flag indicating that a zoom factor will be specified. This flag *must* be followed by a space.
- zoom** Is the zoom factor as a percentage of the image's size. The default is 100 (same size). The zoom factor can be any number greater than 0, such as 25 (¼ size), or 200 (x2 size).
- /F** Specifies that the image is to fit into the size of the image display window. If specified, this means that if the user changes the image display window size, the image will automatically be scaled to fit as best as possible.

Displaying Multiple Images

You can display more than one image on the same screen at the same time. To achieve this, simply send another AiF sequence for the next image, changing the scale as required (and before waiting on input for the response).

There is no limit to the number of images that can be displayed in this manner, simply repeat the AiF sequence for each image. It is often useful to decrease the size of these images by setting the 'scale' parameter to 50 (half size) or less.

Please note that multiple images can only be displayed on the same screen if they *all* have *identical* palettes. Images with different palettes will 'corrupt' each others' screen image (often giving an "infra-red" like display).

Users displaying images will see the images displayed using the resolution and colours as per the current Windows desktop.

Closing the Image Application

This AiF sequence closes a Windows application and should be used to close Windows that are displaying images.

This sequence should be used with great care. Close Application is only intended for use with Windows applications that *do not* support DDE. If an application supports DDE, we strongly recommend that you use a DDE link to close the application.

**ESC_x AP ESC **

Where **_x** is AiF code, and **AP** is the name of the Windows application that is to be closed.

Note: this name should exactly match the name displayed in the application's title bar. This name is not case sensitive but it is sensitive to other factors, such as double spaces, curly brackets, etc.

Closing the Image Application Example

To close the image opened in the previous example, you could use the following sequence:

**ESC_x United Kingdom ESC **

Note: The host application should close down image windows when appropriate. Please bear in mind though, that the user also has this capability. Images may be removed from the user's desktop by using the Close Image Windows Application sequence (described above). Make sure that you specify the correct image window by using the exact name of the window, which will be either "Image [filename]" or "title", if a title was specified when the image was invoked.

Control State of Window

This AiF sequence provides control over the window state (Minimise, Maximise etc.) of a given application already running on the Windows desktop.

**ESC _ ST c AP ESC **

Where:

- ST** Changes the State of the application's window as follows:
- 1 – Activates and displays the window.
 - 2 – Activates and minimises the window.
 - 3 – Activates and maximises the window.
 - 7 – Displays the window minimised but does not change active window.
- c** Is lowercase c - AiF code.
- AP** Is the name of the Windows application. Normally this is the name held in the window Title as shown exactly on the desktop, i.e. WordPerfect [DOCUMENT1 – UNMODIFIED] If AP is null, the state change will affect the current window.

Control Window State Response Format

The response will be

<STX> <CR> status <CR>

Where:

- <STX>** Is the start of text character (ASCII decimal value 002).
- <CR>** Is a carriage return (ASCII decimal value 013).
- status** Is the task number which is 0 if the application does not exist.

Note: It is recommended that you use DDE if it is available and practical because using the server name is more accurate and reliable than depending on the Application Name.

Start Windows Program

Use this sequence to allow any Windows program to be started on the desktop:

**ESC _ ST e PN ESC **

Where:

- ST** Is the state in which you want the Windows program to be started:
- 1 Activates and displays the window.
 - 2 Activates and minimises the window.
 - 3 Activates and maximises the window.
 - 7 Displays the window minimised but does not change active window.
- e** Is lowercase e - AiF code.
- PN** Is the name of the Windows program that you wish to start e.g. 123W or D:\123W\123W (If .EXE is omitted, it is assumed). If you do not specify a drive and/or path, the Windows application will be searched for in the following sequence:
- 1 Look in current directory.
 - 2 Look in the Windows directory.
 - 3 Look in the Windows SYSTEM directory.
 - 4 Look in the directories specified in the PATH variable
 - 5 Look in the directories mapped in a network.

You can also specify startup parameters here. For instance:

d:\123W\123W DEMO.WK3

to start 123 for Windows and open DEMO.WK3 worksheet.

Start Windows Program Response Format

The response will be:

<STX> <CR> status <CR>

Where:

- <STX>** Is the start of text character (ASCII decimal value 002).
- <CR>** Is a carriage return (ASCII decimal value 013).
- status** Is the task number of the application. If it is < 32 the application was not started.

Detect if Windows Application Running

Use this sequence to ascertain whether or not a Windows application is running:

**ESC _ a AP ESC **

Where:

- a** Is lowercase a - AiF code.
- AP** Must be the name of the Windows application. Normally this is the name held in the Window Title as shown exactly on the desktop, e.g. WORDPERFECT [DOCUMENT1 – UNMODIFIED].

Detect Windows Application Response Format

The response will be

<STX> <CR> status <CR>

Where:

- <STX>** Is the start of text character (ASCII decimal value 002).
- <CR>** Is a carriage return (ASCII decimal value 013).
- status** Is the task number which is 0 if the application is not running on the desktop.

Note If the application supports being a DDE server, it is recommended that you use the ESC_ld SN;TP ESC \ sequence (Initiate DDE) to detect whether that server application is active or not, because using the server name is more accurate and reliable than depending on Application Name.

Send Keys to Windows Applications

This sequence sends keys in the DOS keyboard stacker format to the specified Windows application AP. This allows almost any Windows application to be driven automatically.

There is no conversation taking place between HostAccess and the Windows product to which you are sending keys, so you have no way of validating that the keys have been accepted by the other product. We recommend that you at least validate that the other application is running before you attempt to send keys. To do this, you can use either the AiF sequence Detect if Windows Application is Running (described earlier), or the sequence Initiate DDE see Dynamic Data Interchange.

**ESC _ k AP % keys ESC **

Where:

k Is lowercase k - AiF code.

AP Must be the name of the Windows application. Normally this is the name held in the window Title as shown exactly on the desktop, e.g.

WORDPERFECT [DOCUMENT1 - UNMODIFIED]

If AP is null, keys are sent to the current window.

% Is an AiF delimiter.

keys In the same format as the DOS keyboard stacker (see DOS Keyboard Stacker on page 184 for details).

You should use DDE if it is available and practical, as using the server name is more accurate and reliable than using the Application Name.

Miscellaneous AiF Facilities

The following topics document several of the miscellaneous but often very important features within AiF.

For additional or modified AiF features, please contact your dealer or Rogue Wave directly. We have a policy of incorporating user feedback directly into future releases where these requests fall within the general development strategy. (For details of the latest enhancements available in HostAccess, refer to the READ.ME file on the HostAccess disk).

Closing HostAccess From Host

An AiF sequence is available to close HostAccess. This feature allows application developers to include 'close HostAccess' in their application menus.

HostAccess will NOT ask the user to confirm the close request as would be done if ALT/X was entered from the keyboard.

The user will be returned straight to DOS and without any warning if the HostAccess parameters have been changed.

Closing HostAccess From the Host AiF Sequence

Use the following AiF sequence:

```
ESC _ X ESC \
```

Notes: After closing HostAccess the application should normally close the host process that was driving HostAccess.

This AiF sequence is often used in conjunction with automated File Transfer to or from the host but initiated from the PC through the use of HostAccess's Macro Language. It enables the PC to process file transfer(s) remotely, logoff the host session and then exit HostAccess to return to the controlling DOS process (batch file).

For more information on the macro language, please see Chapter 5 - Using the Macro Language.

Getting HostAccess Run-time Status

This AiF sequence may be used to find out more information about HostAccess and its run-time environment.

Use the following sequence to get the HostAccess Run-time status:

ESC [= 10 n

This returns:

<STX> <CR> a;b;c;d <CR>

Where:

- | | |
|--------------------|---|
| <STX> | Is the start of text character (ASCII decimal value 002). |
| <CR> | Is a carriage return (ASCII decimal value 013). |
| a | 1 if Windows version running or 0 if DOS version. |
| b | 1 if current PC is colour, 0 if mono. |
| c | 1 if blinking is enabled on PC, 0 if not. |
| d | 1 if this PC has a mouse that HostAccess can use, 0 if not. |

Note This sequence combines several AiF sequences into one. In time, we will extend the information returned by this sequence, and for this reason we recommend that developers use this sequence in preference to the individual sequences to get serial number, blinking status, etc.

Capturing Screen Text

As well as being able to dump the current screen text to an attached local printer or to a DOS file, you can also send that same screen text up to the host. This screen text can then be sent on to the system printer, saved in a file or indeed used anywhere else on the host system.

Upon sending the appropriate sequence, the PC will send the screen text back to the host. Each line of the screen is sent to the host with all non-printable characters replaced by spaces and terminated by a carriage return.

Capturing Screen Text AiF Sequence

Use the following AiF sequences to capture screen text.

ESC [= 2 i

or

ESC [2 ; n i

Where:

- n** Is the optional parameter determining which screen is sent to the host, as:
HostAccess returns the screen to the host with each line separated by a carriage return and adds a leading and trailing start of text (ASCII value 002) character. The format of the reply to this AiF sequence is:

```
<STX> <CR> line1 <CR> line2 <CR> ... lineN <CR> <STX> <CR>
```

Where:

<STX> Is the start of text character (ASCII decimal value 002).

<CR> Is a carriage return (ASCII decimal value 013).

line1 ... lineN Is each line of the screen. The number of lines will vary depending upon the current screen configuration.

Capturing Screen Text Example

As each line of the screen is terminated by carriage return, a simple program can be written to retrieve each line of the screen image into an array.

For example:

```
screen = ""
counter = 1

send to PC 'ESC[=2i'
echo off
loop input resp until resp equals STX do repeat
loop
  input line
until line equals STX do
  screen(counter) = line
  counter = counter + 1
repeat
echo on
display counter:"screen lines sent to host"
```

Notes: Host echoing of terminal input must be switched off before requesting the screen image. Otherwise, the user's application screen will be corrupted.

This feature can be very useful for documenting applications screens as well as giving users the ability to capture any screen at any time back to the host system.

Another method of (automatically) sending the host session's screen back to the host would be to:

1. Assign the DOS Print device to a DOS file name.
2. Send the ANSI sequence to "print screen"
3. File transfer the DOS file up to the host.

This method may be simpler in some circumstances and would enable you to capture IBM graphics within the screen.

Changing Emulation

In some applications areas it may be useful to be able to change the current terminal type that HostAccess is emulating. A special AiF sequence is provided for this.

ESC [= n {

Where:

n is the emulation number for the required emulation as follows:

0	VT100	11	SM 9400
1	VT220 (7 bit)	12	Ansi
2	VT220 (8 bit)	13	Videotex
3	Prism8/9	14	Microfusion
4	Prism9 Ansi	15	Ampex
5	QVT119	16	TV1920
6	Wyse50	17	Galileo
7	Wyse60	{	The literal character '{'.
8	AddsVp		
9	UCL Term		
10	DG 216		

Changing Emulation Example

Applications which invoke other applications specifically enhanced for different terminals can now swap between the required emulation. In practice, this only occurs very rarely. However, we do know of one application that was built around one terminal type but has later been enhanced to call another "word-processing" application that was specifically targeted for a different terminal type.

Notes: Changing emulations will effectively reset the terminal, wiping out all previous backpages, screens, slots etc. If the previous environment needs to be saved, use the AiF sequence to "push environment", in Save Environment on page 147.

File Transfer

Use the following AiF sequence to start a file transfer:.

```
ESC _mode ; hostdriven ; 1 ; append ; 0 ; protocol ; ist ; direction local ;
Remote { ; FTP server } { ; username } { ; password } ESC \
```

Where:

Mode	0 = binary. 1 = text.
hostdriven	0 = Displays progress dialog during the transfer. User must close the dialog manually once the transfer is complete. 1 = Displays progress dialog during the transfer and automatically closes the dialog once the transfer is complete (DOS.PICK Flag = H). 2 = Suppress all progress output (DOS.PICK Flag = Z).
append	0 = Overwrite destination file. 1 = Append to destination file. Note: If using protocol number 9, this parameter is ignored.
protocol	0 = Proprietary. 1 = Kermit. 2 = X/Ymodem. 3 = Zmodem. 9 = FTP
ist	0 = Transfer is to local PC file (normal). 1 = Intersession file transfer. Note: If using protocol number 9, this parameter is ignored.
direction	{ = Send file to host. } = Receive file from host.
local	Filename on the PC.
Remote	Filename on the host.
ftp server	The FTP server address.

	Note: Only relevant when using protocol 9.
username	The username to be used when connecting to the FTP server.
	Note: Only relevant when using protocol 9.
password	The password for the username above
	Note: Only relevant when using protocol 9.

For example:

```
ESC_1;0;1;0;0;3;0{c:\monkey.txt;pigESC\
```

Will start a Zmodem file transfer to send the file 'c:\monkey.txt' on the PC to the file 'pig' on the host.

FTP example:

```
ESC_0;2;1;0;0;9;0;}c:\dn\drivers.zip;/services/technet/drivers.zip;ftp.microsoft.com;anonymous;  
passwordESC\
```

Will download the binary file '/services/technet/drivers.zip' from the ftp.microsoft.com ftp server into the local file 'c:\dn\drivers.zip'. No progress dialog will be displayed.

Dynamic Data Exchange

The following topics how DDE works and summarise the DDE Escape sequences. They explain how you can use DDE with HostAccess, DDE Client support and DDE Server support.

How DDE Works

Dynamic Data Exchange (DDE) is used to transfer data between Windows applications.

Two applications that participate in DDE engage in what is known as a DDE **conversation**. The application that initiates the conversation is known as the **client** application, and the application that responds to the client is known as the **server** application.

Any Windows product that supports DDE as a server application must have a **server name**. For example, the server name for Word for Windows is WINWORD and for Quattro Pro it is QPW. To initiate a DDE link with a server application, you would normally use the server name and this would return a **channel number**. Using this channel number you would then send commands (normally in the format of the macro language supported by that server application) and finally close the link with that channel number when all processing is completed.

When communicating with a server you must also always specify a **topic**. Server applications can support many topics depending on which part of that application you want to communicate with. For instance, if you want to request information from Quattro Pro on a specific spreadsheet, the server name would be QPW and the topic name would be the spreadsheet name.

DDE was designed to form a standard way of communicating between Windows applications. However, the fact that each Windows application supports DDE differently (or sometimes not at all) makes it more difficult for the novice to understand it or become involved with it.

If you want to program using DDE, you will have to learn as much, if not more about the server application that you want to talk to, rather than if you were a direct user of the product itself.

DDE Sequences: Summary

<code>ESC _9d SN;TP ESC \</code>	Close a DDE link already established with Initiate DDE sequence.
<code>ESC _ 2 ; TM d SN ; TP ; MA ESC \</code>	Send commands to server application.
<code>ESC _ 1d SN;TP ESC \</code>	Open a DDE channel with a server.
<code>ESC _ 3; TM d SN;TP;IT;ST ESC \</code>	Pass data to server.
<code>ESC _ 4; TM d SN;TP;IT ESC \</code>	Retrieve data from server.

Using DDE with HostAccess

You can use DDE to use HostAccess as a **DDE client** to other Windows applications (servers), sending data and instructions from the host to a Windows application. This gives your own host programs and applications almost total control over any other Windows product.

Using DDE with HostAccess, you only need to specify the server name for any DDE process. HostAccess automatically keeps track of channel numbers internally for you.

All Windows applications support a general topic name **system**. Unless you are setting up more complicated DDE links, this topic should be more than adequate for most developers. (*All* of the HostAccess PASS.TO's for Windows use the SYSTEM topic).

You can also use DDE to use HostAccess as a DDE server. You can write Windows programs in such applications as Word or Excel, which can send and receive data to and from the host software.

Note: You must have a resilient link from the PC to the host. DDE cannot work remotely unless full flow control and error checking are in place.

DDE Client Support

The following topics describe the AiF sequences used to connect, communicate, and disconnect between client and server applications in a DDE environment.

You should have a full knowledge of DDE before using these features.

Initiating a DDE Conversation

To open a DDE channel with a Windows application, use the following AiF sequence:

```
ESC _ 1d Server ; Topic ESC \
```

Where:

Server Is the Server name of the application.

Topic Is the Topic for that application.

If a link is already open to this server and topic it will be re-used.

DDE Response Format

The DDE initiate response will be:

```
<STX> <CR> status <CR>
```

Where:

<STX> Is the start of text character (ASCII decimal value 002).

<CR> Is a carriage return (ASCII decimal value 013).

status Is 0 if successful and > 0 otherwise, as follows:

- 1 - Application link not open (no initiate).
- 2 - Timeout.
- 3 - Topic not supported by application.
- 4 - No DDE channels available.
- 5 - Server closed.
- 6 - Server busy.
- 7 - Server NAK (Not Acknowledge).

Sending Commands to the Server

To send commands to the server application (to allow it to be driven and updated automatically), use the following sequence:

**ESC _ 2 ; Timeout d Server ; Topic ; Mstring ESC **

Where:

Timeout If used sets the timeout on DDE commands to that number of seconds. If a DDE macro being passed is going to take a long time it is sometimes worth using a high value to stop the DDE terminating on a timeout (NAK).

Server Is the Server name.

Topic Is the Topic name, normally SYSTEM.

Mstring Is the macro string in the format expected by the DDE server application.

A successful DDE Initiate *must* have been made with Server name and Topic before the macros can be sent.

For example:

[FileOpen.Name = "C:\WINWORD\TEST.DOC"][[FileOpenDataFile.Name etc.]]

or

{FileOpen C:\QPWTEST.WB1}{COLUMNWIDTH A1..C20,1,2,3}{etc.,}

Multiple macro commands may be passed using the above square brackets to separate each macro command. Some products (like Quattro Pro) seem to prefer curly square brackets rather than normal square brackets like most other Windows applications. Please check the documentation for the Server Application if normal square brackets do not work.

Response Format

The DDE initiate response will be

<STX> <CR> status <CR>

Sending Data to a Server (Poke)

This sequence allows data to be passed directly to another Windows application (the server). Most DDE servers have defined elements (**items**) that the server knows about, which can accept data from DDE clients. For example, R1C1 is the item name for some spreadsheet packages.

A successful DDE Initiate must have been made with Server name and Topic before the data can be sent.

**ESC _ 3; Timeout d Server ; Topic ; Item ; String ESC **

Timeout If used sets the timeout on DDE commands to that number of seconds. If a DDE macro being passed is going to take a long time it is sometimes worth using a high value to stop the DDE terminating on a timeout (NAK).

Server Is the Server name.

Topic Is the Topic name, normally SYSTEM.

Mstring Is the macro string in the format expected by the DDE server application.

Item Is the Item name recognised by the DDE server.

String Is the string of data to put into IT specified above.

The response will be:

<STX> <CR> status <CR>

Requesting Data from a Server

This sequence allows data to be retrieved directly from another Windows application (the server). Most DDE servers have defined elements which the server knows about (called **items**) where specific pieces of information reside.

A successful DDE Initiate must have been made with Server name and Topic before the data can be retrieved.

**ESC _ 4; Timeout d Server ; Topic ; Item ESC **

where **Timeout**, **Server**, **Topic** and **Item** are as defined above.

The response will be

<STX> <CR> status <CR> string <CR>

string is the data held as returned from the Server application. The data may be tabbed or comma delimited, dependent on the server application.

Close DDE Link

This sequence closes a DDE link already established with the Initiate DDE sequence. It is recommended that you close DDE links when any DDE conversation is completed.

**ESC _9d SN;TP ESC **

Where

SN Is the Server name.

TP Is the TOPIC of the DDE session to close conversation with, normally SYSTEM.

DDE Server Support

You can use HostAccess to act as a DDE server to Windows applications such as Word and Excel, sending data to the host, receiving data from a host and allowing the Word or Excel application to obtain the results.

Any Windows product that supports DDE as a server application must have a **server name**. In HostAccess's case this is:

Servername: HA7

The following operations are supported:

DDE Requests:

Topic		System
Items	Topics	Returns a list of available topics
	Formats	Returns a list of supported formats
	SysItems	Returns a list of supported items for this topic
Topic		Session name
Items	CursorPos	Returns the current cursor location
	ScreenSize	Returns the current screen size
	RyCxNc	Returns text from screen (RxCxNx) where R=row, C=column, N=no. of characters to read
	Items	Returns a list of supported items for this topic

DDE Execute:

Topic		Session name
Items	<Empty>	Executes a macro or macro command. To execute a macro, supply the path to the macro file. To execute a macro command prefix the command with * e.g. *PRINT "Hello World"

DDE Poke:

Topic		Session name
Items	Keys	Send keys as if they were typed on the keyboard. These are in the mnemonic format as described in the mnemonic key code table e.g. CR for carriage return, ES for ESCape, F1 for function key 1, etc.
	Network	Send data to the host

DDE Example

The following WordBasic example provides a list of the available system items, topics and formats supported by the application:

```
Sub MAIN
  DDETerminateAll
  n = DDEInitiate("HA7", "system")
  a$ = DDERequest$(n, "SysItems")
  MsgBox(a$, "SysItems")
  b$ = DDERequest$(n, "Topics")
  MsgBox(b$, "Topics")
  c$ = DDERequest$(n, "Formats")
  MsgBox(c$, "Formats")
  DDETerminate n
End Sub
```

Some applications that support DDE: Microsoft Word for Windows, Excel, Visual Basic.

Using the Macro Language

The HostAccess macro language is a simple and powerful tool that allows you to automate standard tasks. For example, you can use the macro language to automate your login procedure, or to call a Windows application and run a set of tasks within it.

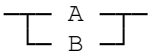
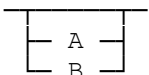
This chapter describes all the features provided in the macro language, and how to use each one.

The example provided at the end of this chapter demonstrates the power of the macro language.

This documentation assumes that you are familiar with basic programming concepts, such as loops, variables, expressions and commands.

Syntax Conventions

Each command described has an associated syntax diagram, to help you understand the exact usage of the command. These diagrams should be intuitively clear. In case of confusion, refer to the following table for explanations of the conventions used:

Symbol	Meaning
>—	Start of a command.
>>	Continuation character.
—><	End of a command.
	New line.
	Either A or B must be chosen.
	Either A or B may be chosen.
<<< — A —	A may be repeated.
<, < — A —	As above, but each repetition must be separated by a comma

Commands are described in upper case (and emphasised in bold in the syntax diagrams).

Variables, **procedures** and **functions** are described in lower case. The macro language itself is case-sensitive for variables, procedures and function names, but not for command statements.

Using AiF Escape Sequences

You can use the macro language to send and receive AiF escape sequences.

AiF escape sequences are normally sent from the host to the PC, and replies are returned from the PC to the host. Under some circumstances, however, you may want to control the operations entirely from the PC; for example, when you have no control over a host program.

Use the PRINT command to send an AiF escape sequence, and the INPUT command to read a reply from the AiF.

Declaring Variables

Before using a variable, you must declare it. To declare a variable, use one of the following statements:

- **DIM**: for local declarations.
- **GLOBAL**: for global declarations. Global variables **must** be declared, and survive between macro programs and HostAccess sessions. Therefore, we recommend you limit the number of global variables declared.

Examples

```
DIM a, b, c AS INTEGER
```

```
GLOBAL name AS STRING
```

Syntax

```

          << , <<<
>— DIM ——— variable — AS — INTEGER —————><
   | GLOBAL |               | STRING |

```

Using Functions

Use a **function** as an expression, which returns a value. Define functions with the **FUNCTION ... END FUNCTION** command, and call them as expressions.

Example

```

FUNCTION squareadd(b AS INTEGER,c AS INTEGER) AS INTEGER
  REM returns the square of two parameters b and c
  squareadd=b*b + c*c
END FUNCTION
:
:
LET a = squareadd(14,6)

```

Syntax

```

>- FUNCTION name _____ AS type | _____>>
    | _____ |
    | ( _____ ) |
    | <<<<<< , <<<<<<< |
    | _____ |
    | variable AS type |
    |
    <<<< , <<<<<
>>- declaration — name = expression _____ | END FUNCTION -><
    | _____ |
    | code |

```

System Functions

Following are descriptions of the system functions available with the macro language.

Name	Purpose	Call As
Chr\$	Converts integer to string character.	Chr\$(n)
Field\$	Returns the nth item from the string list, where each item is separated by the string s. For example, Field\$("Hello;world", ";", 2) = "world".	Field(list,s,n)
Id\$	Used with the INPUT command.	Id\$(string)
Index	Returns the starting character position of string2 within string1. 0 is returned if string2 is not found within string1. For example, Index("hello world", "world") = 7.	Index(string1,string2)
Left\$	Returns the n leftmost characters of string.	Left\$(string,n)
Len	Returns the number of characters in a string.	Len(string)
Lower\$	Returns every character within the string as lower case.	Lower\$(string)
Ltrim\$	Returns string argument without leading spaces.	Ltrim\$(string)
Mid\$	Returns (m characters of) the string from character n onwards.	Mid\$(string,n) or Mid\$(string,n,m)
Reply\$	Used with the INPUT command.	Reply\$(string)
Right\$	Returns the n rightmost characters of string.	Right\$(string,n)
Rtrim\$	Returns string argument without trailing spaces.	Rtrim\$(string)
Screen\$	Returns the word found from the screen at x,y. If x is set to 0 the whole line will be returned.	Screen\$(x,y)
Trim\$	Returns string argument without trailing or leading spaces, and collapses multiple spaces.	Trim\$(string)
Upper\$	Returns every character within the string as upper case.	Upper\$(string)
Val	Returns the numeric value of a character string.	Val (string)

Name	Purpose	Call As
Waitkey\$	Waits for user to press a key, then returns that key as a string.	Waitkey\$

Using Procedures

Call **procedures** to perform specific discrete actions, and then return to the calling point in the program. Define procedures with the SUB ... END SUB command, and call them using the CALL command.

To terminate a procedure (for example, on an error), use the EXIT SUB keyword. This returns control to the calling program.

Example

```
SUB greet( name AS STRING )
    PRINT "Hello World from";name
END SUB
:
:
CALL greet ("David B.")
```

Syntax

```
<<<< , <<<<<<
>- SUB name |-----| declaration |----->>
    |-----|
    | ( |-----| ) |
    | <<<<<< , <<<<<< |
    | variable AS type |
```

```
>>----->>
|-----| | END SUB |----->>
| code | | EXIT SUB | | code |
```

Macro summary

Command	Description	Example
CALL	Calls a (previously-defined) procedure.	CALL SubProcedure ()
DELAY, DELAYTILL	Delays a set number of seconds, or until a specified time.	DELAY 5
DIM	Declares a variable as INTEGER or REAL.	DIM a AS INTEGER.
DO (WHILE) ... LOOP	Starts a program loop, continuing while the WHILE condition holds, exited when WHILE condition is fulfilled.	LET A=10 DO WHILE A>=2 LET A=A-1 PRINT A LOOP
END	Stops a macro.	
EXIT	Exits from the current loop or IF statement. For example, to exit a FOR loop.	EXIT FOR
FOR ... NEXT	Creates a loop of a specific duration.	FOR i = 1 TO 10
GOTO	Transfers control to a part of a program with a pre-defined label.	L20: : PRINT A GOTO L20
IF ... THEN ... ELSEIF	Specifies one or more actions to take if a condition is fulfilled.	
LET	Assigns a value to a variable. Variables must be declared with DIM before being assigned.	DIM A AS INTEGER LET A = 5.
PASSKEYS	Suspends macro processing to allow the user to enter keystrokes to the host.	PASSKEYS
PASSKEYSNOCR	Suspends macro processing to allow the user to enter keystrokes to the host – does not send the final Enter to the host	PASSKEYSNOCR
PRINT	Prints a text message to the Host, or the session screen, or to the status bar.	PRINT "HELLO"
REM	Used for code comments.	REM This will automatically log REM you onto a host.

SELECT Selects alternative actions based on specified conditions.

Command	Description	Example
SENDTERM	Sends text to the host.	SENDTERM PASSWORD, CHR\$(13)
SEND, SENDWIN	Sends special characters to the host, or to the currently-active Windows application.	
WAIT (TIMEOUT)	Waits for a host response (optionally, for a maximum timeout period).	WAIT TIMEOUT 20
WHILE ... WEND	Specifies a loop containing one or more instructions to be carried out whilst a condition holds.	<pre>DIM B AS INTEGER WHILE B >=1 PRINT B LET B = B-1 WEND</pre>

CALL

Use this command to call a previously-defined procedure. Depending on the procedure, the call may pass parameters to the procedure.

See page 218 for more details on using procedures.

Syntax

```
>- CALL subname _____><
      |<<<< , <<< |
      | expression |
```

DELAY

Use the DELAY command to insert a delay into your programs of a specified number of seconds. The following example will insert a delay of 20 seconds.

Example

```
DELAY 20
```

Syntax

```
>- DELAY seconds _____><
```

DELAYTILL

Use the DELAYTILL command to insert a delay into your program, until a specified time. The following example will delay the macro processing until 10:30.

Example

```
DELAYTILL 10:30
```

Syntax

```
>- DELAYTILL hh:mm _____><
```

DO

This command allows you to create a program loop. This loop will end when the pre-specified condition is fulfilled. Use the EXIT DO keyword to exit the loop early (for example, on error).

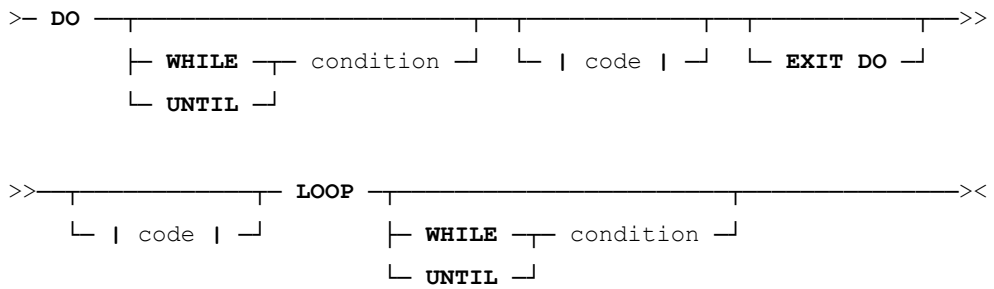
Example 1

```
LET a=10
DO WHILE a>=2
    LET a=a-1
    PRINT a
LOOP
```

Example 2

```
LET a=1
DO
    LET a=a+1
    PRINT a
LOOP UNTIL a>10
```

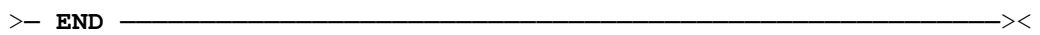
Syntax



END

Use this command to stop your programs (under normal circumstances).

Syntax



EXIT

Use this command within a loop (whether FOR, WHILE, DO, SELECT or WAIT), to exit from the current loop in your program.

Alternatively, you can add the name of the loop as a qualifier, for example EXIT FOR.

Example

```
DIM i as integer
:
:
FOR i = 0 TO 10
  IF a = 0 THEN
    REM At end of list, jump to after NEXT loop
    EXIT
  ENDIF
NEXT i
```

Syntax

>- EXIT -----><

INPUT

Use this command to input an AiF reply into a string variable. The format of the reply will be:

<STX>n<CR>message<CR>

Use the **Reply\$()** function to read **message**, and the **id\$()** function to read **n**. These components depend on the AiF escape sequence sent, and the reply received. Refer to Chapters 3 and 4 for details of AiF escape sequences available.

Example

This example prints (sends) an AiF escape sequence determined by the string **sequence\$**, then reads the reply into the string **return\$**. It then reads the **message** component of the reply into the string **msg\$**.

In this example, the AiF sequence sent asks HostAccess for its version number, then puts the reply into a string, and prints this information to the screen.

```
DIM Sequence$ as STRING
DIM msg$ as STRING
DIM return$ as STRING
:
:
Sequence$ = CHR$(27) + "[=1c"
Print sequence$      : REM send AiF sequence
REM
Input return$        : REM read reply
msg$ = reply$(return$)
PRINT "Version Number is: " + msg$
```

Note: **sequence\$**, **return\$** and **msg\$** need to be pre-defined via the DIM command as shown. **Reply\$** is a system function, see page 216.

Syntax

>—INPUT string—————><LET

LET

Use this command to assign a value to a variable.

Example

```
LET a = 10
```

Syntax

```
> _____ variable = expression _____><
  |_____|
  |  LET  |
```

PASSKEYS

Use this command to suspend macro processing, to allow the user to enter keystrokes be passed to the host. The user keyboard input is passed until the Enter key is pressed. This Enter key is passed to the host as a carriage return and the macro the resumes processing at the next line.

For example, a “login” macro can effectively pause whilst a user types in a unique password (not held within the macro) and then continue with invoking the user’s host application.

Syntax

```
>- PASSKEYS _____><
```

PASSKEYSNOCR

Use this command to suspend macro processing, to allow the user to enter keystrokes be passed to the host. The user keyboard input is passed until the Enter key is pressed. This Enter key is not passed to the host and the macro the resumes processing at the next line.

Syntax

```
>- PASSKEYSNOCR _____><
```

PRINT

Use this command to print out a text message, to the Host, or the session screen, or to the status bar.

The PRINT command moves onto the next line when finished, unless the PRINT statement ends in a semi-colon (“;”) or comma (“,”) character.

Use the comma character to separate printed items with a tab.

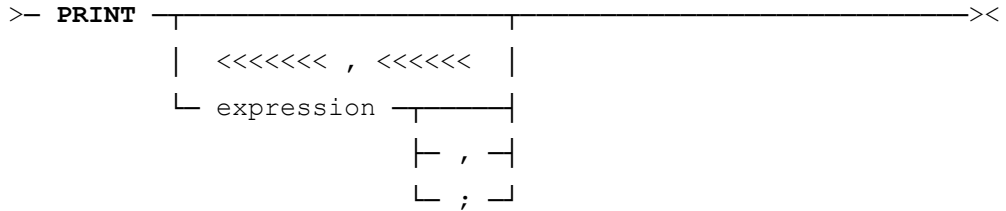
Use the semi-colon character to print items with no spacing.

Use the plus character to print items with no spacing.

Example

```
PRINT "Hello" , "O" , "There"
PRINT "Hello" ; "O" ; "There"
PRINT "Hello" + "O" + "There"
```

Syntax



REM

Use this command to put comments in your code.

Syntax

```
>- REM remark_____><
```

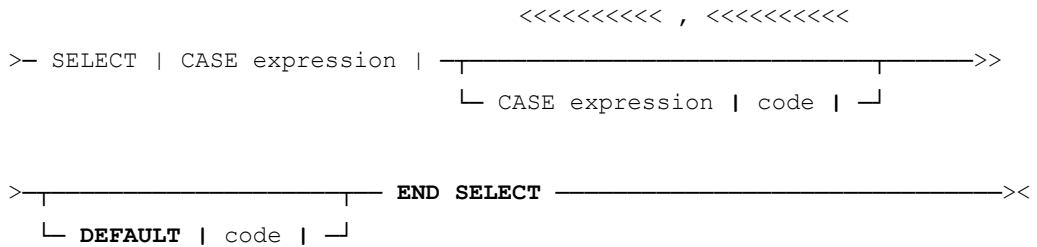
SELECT

Use this command to define alternative actions based on specified conditions more easily than with the IF command.

Use the DEFAULT keyword to specify the default action to be taken (if no conditions are met).

Note: you can use the EXIT SELECT keyword to exit from the SELECT construct.

Syntax



SENDTERM

Use this command to send text to the host.

Syntax

```

<<<<<<<
>- SENDTERM string -----><

```

SEND and SENDWIN

Use the SEND command to send special character codes to the host. This sends data to the host as though the user had pressed the keys on the keyboard. So, if the user reprograms the '1' key to send 'Hello' then the sequence SEND "1" would send the string "Hello" to the host and not a "1".

Use the SENDWIN command to send special character codes to the currently-active Windows application.

Example

```
SEND ``Hello There' CR"
```

Syntax

```

<<<<<<<
>- SEND -----><
  | SENDWIN |

```

Where **keycode** is one of the mnemonics or special characters defined below:

Mnemonic	Represents	Character	Represents
LA	Left Arrow	^	Control function.
RA	Right Arrow	@	Alt function.
UA	Up Arrow	#	Shift function.
DA	Down Arrow	F	Function key.
PU	Page Up	S	Shift + function key.
PD	Page Down	C	Control + Function key.
Mnemonic	Represents	Character	Represents
HM	Home	A	Alt + Function key.

EN	End	Wnn	Wait time in 55 millisecond units (clock ticks, about 18.2 per second), where nn is from 1 to 255.
IN	Insert	WP	Wait for user key and then pass it on.
DE	Delete	WE	Wait for user key and then throw it away.
TA / TB	Tab	WB	Wait until key buffer is empty.
ST / BT	Shift Tab (=Back Tab)	BR	Break.
ES	Escape	'text'	Enclose literal text within single quotes.
BS	Backspace		
SP	Space bar		
CR	Enter		
LF	Ctrl-Enter		
DQ	The double quote "		
SQ	The single quote '		

WAIT

Use this command to wait for a host response.

Use the TIMEOUT keyword to specify a maximum wait period, and then the same keyword (after a CASE keyword) to specify any action to be taken after that period has passed.

Use the CASE keyword to specify the action to be taken if a specified string is seen.

Note: You can use the EXIT WAIT keyword to exit from the WAIT loop.

Syntax

```

>- WAIT _____>>
      |-----| |-----|
      | TIMEOUT seconds | | x,y |
>- <<<<<<<<<< , <<<<<<<<<<
      < , <<
>- CASE string | code | _____ END WAIT _____>>
      |-----| |-----|
      | TIMEOUT | code |

```

Example

```

DO
  WAIT TIMEOUT 30
  CASE "password:"
    PASSKEYS          : REM allow user to enter password
  CASE "login:"
    SENDTERM "David"  : REM enter a login name
  CASE "$ "           : REM end script at the prompt
  END
  CASE "(ansi)"       : REM Send terminal type
    SENDTERM "VT100"
  TIMEOUT
  PRINT "Give up"
END WAIT
LOOP

```


WHILE

Use this command to specify a loop containing one or more instructions to be carried out whilst a condition holds.

Use the EXIT WHILE keyword to exit the loop early (for example, on error).

Example

```
DIM t AS Integer
LET b = 10
WHILE b >=1
    PRINT b
    LET b = b -1
WEND
```

Syntax

```
>- WHILE condition ----->
      | code | | EXIT WHILE | | code |
-----><
>- | WEND -----><
```

Macro Example

The following example demonstrates the power of the macro language. This is an auto-login facility, which will automatically log you into a Unix machine, given the correct password.

```

DIM retries AS Integer
DIM psw$ AS String

DIM i AS Integer
DIM s AS String

FOR retries = 1 TO 3
  SEND "'root' CR" : REM send user name to host
  WAIT TIMEOUT 10 : REM wait 10 seconds
  CASE "Password:"
    s= Chr$(27) + "_" + 5 + ";" + 10 + ";" + 16 + ";" + 16 +
      ";1;0;0J" + ";" + "Password:" + Chr$(27) + "\"
    PRINT s;
    INPUT psw$ : REM read user reply (password)
    psw$= reply$( psw$ )
    SENDTERM psw$,chr$(13) : REM send password to host
  DEFAULT
    PRINT "No idea. Any clues?"
  EXIT FOR
END WAIT
REM if host asks for terminal type, reply with "vt220"
WAIT TIMEOUT 4
CASE "(ansi) "
  SEND "'vt220' CR"
DEFAULT
  EXIT FOR
END WAIT

WAIT TIMEOUT 4 : REM wait for host to print a prompt
CASE "# "
  EXIT FOR
END WAIT
NEXT : REM And try again

IF retries >= 3 THEN
  PRINT CHR$(27) + "_X" + CHR$(27) + "\";
  REM terminate HostAccess after 3 tries
ENDIF

```



Describing Images

When using Windows AiF escape sequences that refer to buttons, you can describe button images for a particular button in great detail.

- You can define images to suit your requirements.
- You can store image definitions in your (user's) `termw.ini` file, for widespread or repeated use in your applications programs.
- You can display bitmaps or icons, either as separate files (`.bmp` or `.ico` files) or as part of a resource (a `.exe` or `.dll` file).

The facilities described here give extensive and powerful tools to customise your display. However, you can use the most simple features, to display images quickly.

Image Types

There are three ways of referring to images:

- Simple images.
- Images with labels attached.
- Images with button window descriptions.

How to Describe Images

To describe images in a Windows AiF escape sequence, use an **image specification string**. This can be embedded within an AiF escape sequence, or can be pre-defined in the user's `ha6.ini` file (for the 16 bit product) and `ha7.ini` file (for the 32 bit product).

An image specification string contains several parameters, separated by commas. Each parameter takes the form:

name = value

where **name** is the parameter name and **value** is the value for that name. For example, `filename=c:\images\helpbut.bmp` is a parameter.

As each parameter is named, parameter order is unimportant, although we recommend that you follow the documented order for clarity and ease of use.

Conventions Used

In the following topics, image specification strings are described as follows:

{parameter1},{parameter2}, ... , {parameterN}

Where **parameter1 ... parameterN** are the names of the parameters within the string. Optional parameters are enclosed in braces.

Many parameter names and values can be abbreviated in use. For example, **filename** can be abbreviated to **f**, and **bmp** can be abbreviated to **b**. These abbreviations are shown in parentheses. We recommend you only abbreviate after developing and testing your code, to increase readability during development.

Pre-defining Images

You can define and store a set of image specification strings, with labels, for your own images, in the user's ini file. You can then access these images from an AiF sequence, using the images labels.

To create a labelled image:

1. Define an image specification string for your image.
2. Label this string.
3. Place the labelled string your **HA6.ini** file (for the 16 bit product) and **ha7.ini** file (for the 32 bit product) in the HostAccess directory.
4. Store simple bitmap image strings in the [dibs] section.
5. Store strings for bitmap images with labels in the [images] section.
6. Store strings for bitmap images with button window descriptions in the [buttons] section.

This labelling feature simplifies image use within Windows AiF escape sequences. Instead of hard-coding image specification strings directly into the AiF sequence, you can simply refer to their label in the .ini file.

We recommend that you make full use of this facility, for any but the most simple specification strings.

HostAccess also has a series of pre-defined images for you to use - these images are described in the following sections.

Using Named Images

To use a named image from an AiF sequence, refer to it by name.

To use a named image containing a label or a button window description from an AiF sequence, precede the name with a @ character.

Example

The following section could be in a typical .ini file:

[images]

frog = filetype=bmp, filename=frog, su = 1-black

To use this image, you can then refer to it as @frog. For example, you could use:

ESC_31 ; 10 ; 10 ; 5 ; 5 w animal ; @frog ESC

This gives a 5x5 push-button called **animal** at (10,10), using the string labelled **frog** in the [images] section of the user's host.ini file.

Defining a simple image

To specify a simple image, use a specification string as follows:

{filename},{filetype},{id},{tilesize},{subst}

Where:

Name	Description
filename (f)	File name of the image. A default file extension is added, depending on the file type (see below). There is no default file name. You must give this parameter, except when accessing an internal resource in HOST.EXE. The default file location is the current working directory. To specify another location, give the full path name.
filetype (ft)	File type. Can be any of the following: <ul style="list-style-type: none"> bmp (b) - .bmp file (the default). bmpexe (be) - bitmap in a .exe or .dll file. ico (i) - .ico file. icoexe (ie) - icon in .exe or .dll file (i.e., in any file in 'new executable' format).
id	Resource id. This selects the particular icon or bitmap resource from a .exe/.dll file. It may be a name or a number. There is no default for this - it must be given if the bitmap is being taken from a .exe/.dll file.

- tilesize (ts)** Tile size in pixels, given as x/y. Default is 57 by 33, which is the standard size used in dialog boxes for Borland-style bitmap push buttons. e.g. `tilesize = 16/16`.
- subst (su)** Use to substitute colours in an image – typically, to substitute the terminal background colour for the background colour of the bitmaps. Substitution may be repeated.

Example

To use the file 'frog.bmp', in the directory 'c:\pictures', use the following specification string:

```
file=c:\pictures\frog
```

Since no extension was given, **.bmp** is assumed to be the default file type.

To use this string in an AiF escape sequence, use:

```
ESC_31 ; 10 ; 10 ; 5 ; 5 w animal ; file=c:\pictures\frog ESC\
```

This creates and displays a 5x5 push-button named **animal** at (10,10), using our "frog.bmp" file.

Colour Substitution

To substitute a colour, use either:

N-R-G-B

or:

N-name

where **N** is the new (replacement) colour, and **R-G-B** or **name** specifies the old colour. These parameters are described in the following sections.

Specifying the New Colour

N is the new (replacement) colour. This is either a terminal window colour number from 1-16, or is a Windows system colour, taken from the following list:

Name	Colour	Name	Colour
Bg	Desktop background	hi	Highlighted background
Menu	Menu background	hitext	Highlighted text
Win	Window background	btnface (bf)	Button background
Wintext	Window text	btntext (bt)	Button text
Menutext	Menu text	btnshadow (bs)	Button edge colour

Name	Colour	Name	Colour
Appworkspace	Background colour for MDI apps	btnhi	Button highlight colour
		greytext	Grey text colour

Specifying the Old Colour

To specify the colour to be replaced, you can:

- Define the colour in terms of its **R-G-B** (Red-Green-Blue) components, where **R**, **G** and **B** are in the range 0..255. For example, 0-0-255 is full intensity blue, and 0-0-0 is full intensity black
- Use a pre-defined colour **name**, as a shortcut way of specifying common colours:
red (r) blue (b) magenta (m) white (w)
green (g) cyan (c) yellow (y) black (b)

For example, to substitute all black pixels with colour 1 from the application palette, use:

```
su=1-black
```

Note: this colour must **exactly** match the colour of the image.

Examples: colour substitution

To use the file **frog** in your current directory, substituting all black pixels (RGB=0,0,0) with colour 1 from the application palette, use:

```
f=frog,su=1-bl
```

To use a bitmap in a **.exe** format file called 'bitmaps.dll', with resource number 116, tile size 48x64, and replace green pixels (RGB=0,255,0) with the system's button face colour, use:

```
f=bitmaps.dll,ft=be,id=116,ts=48/64,su=btnface-green
```

To display an image based on this string, you could use the following AiF escape sequence:

```
ESC_31 ; 2 ; 2 ; 5 ; 5 w icon1 ; f=bitmaps.dll,ft=be,id=116,ts=48/64,su=btnface-green ESC\
```

This displays a 2x2 image named **icon1** at (5,5), using the string described above. See Chapter 2 for further details of AiF escape sequences.

Inbuilt Images

The following bitmap images are built into HostAccess, and can be used by the host:

Name	Description
<code>_hand</code>	bitmap used to display the open palm image used in warning dialogs.
<code>_applogo</code>	The 'application logo' - the bitmap image for the product, as shown in the about and splash boxes.
<code>_logo</code>	The 'Company logo' - the bitmap image for the Company producing the product, as shown in the About... and splash boxes.
<code>_sp</code>	Standard Push Button images: This is a tiled bitmap image. Tile 1= Cross., Tile 2= Help, Tile 3= No, Tile 4 = Yes/OK
<code>_sm</code>	Standard Message Box images. This is where the large exclamation mark, info symbol and question mark bitmaps are defined. Note that the 'hand' symbol logically belongs to this set, but is in a separate bitmap because it is a different size. Tile 1=info, Tile 2=exclamation, Tile 3=question mark.

You can also define your own built-in bitmaps.

Defining Labelled Images

To include text labels in your image, use the following specification string:

{bitmap/image parameters},{label},{labelpos},{sm},{mag},{tile}

If you have a named bitmap image pre-defined in your host.ini file, use the **bitmap** parameter to refer to it.

If you wish to define a whole image in one specification, use **image parameters**. These are the standard parameters for defining an image.

Name	Value
bitmap (bm)	The label of a pre-defined bitmap image.
image parameters	The filename, filetype, id, tilesize, and subst parameters, as for a simple bitmap specification string.
label (l)	Text for label for the image, drawn at a position offset from the top-left of the output rectangle by the 'labelpos' value.
labelpos (lp)	Label position, given as x/y, used to decide the origin of the start of the text label. Default = (0,0).
sm	Stretch mode. Possible values are: clip (c) clip the image to the destination display rectangle. mag (m) magnify the image as much as possible, whilst retaining aspect ratio. fill (f) stretch/compress the image to fit the destination rectangle exactly. (default)
mag (m)	Magnification factor. A positive or negative integer, controlling the size of an image. A negative number will reduce the size, a positive number will increase the size. For example, specify -2 to divide the size by 2, or 3 to magnify the size by 3. Obviously, -1, 0 and 1 will have no effect.
tile (tl)	This specifies the tile number for the image. Used when the images is tiled, holding an array of separate images of the same size. The first tile is no. 1. The default is not to select a tile - i.e., the whole of the source image.

Specifying Text-Only Labels

To specify text labels (without any bitmap images), use the following image specification string:

type=t,{label},{size}

Name	Value
type (t)	Sets the type: type=t (or type=text) sets image as a text label
label (l)	Text for label. This string will be drawn centered in the output rectangle.
size (s)	Specifies the natural size of the button, in pixels. This defaults to the size of the rectangle needed to exactly hold the label, using whatever font the image is being output with.

Example: Using Pre-defined images

To use an image based on a pre-defined bitmap image named ‘_sp’, with tile 1, label position x=26 y=17, label ‘Cancel’, you could use either:

bitmap=_sp, type=bitmap, tile=1, labelpos=26/17, label=Cancel

or

bm=_sp, tl=1, lp=26/17, l=Cancel

Note: This is the exact definition of the image used in the standard Cancel push button.

To use this image in a 2x4 push-button named **cancel** at (5,5), use the following AiF escape sequence:

ESC_31 ; 2; 4 ; 5 ; 5 w cancel ; bm=_sp, tl=1, lp=26/17, l=Cancel ESC

To use a bitmap image, setting natural size to a magnification of 2, when rendering the image is to be stretched/compressed whilst retaining its aspect ratio, from the file frog.bmp, and substituting all black pixels (RGB = 0,0,0) with colour 1 from the application palette, use:

f=frog.bmp, m=2, sm=mag, ft=bmp, su=1-1-black

To use a text based image (i.e. an image not based on a bitmap), with label ‘Cancel’, and natural size 100x32 pixels, use:

t=t, l=Cancel, s=100/32

This might be used to describe a textual button in a dialog box.

Inbuilt Labelled Images

There are several inbuilt image specifications that you can use:

Name	Description
_cancel	The image and label 'Cancel' to go inside a Borland-style push button.
_help	The image and label 'Help' to go inside a Borland-style push button.
_yes	The image and label 'Yes' to go inside a Borland-style push button.
_no	The image and label 'No' to go inside a Borland-style push button.
_ok	The image and label 'OK' to go inside a Borland-style push button.
_hand	The image to go in a warning message box.
_logo	The image used in the About... and splash dialogs holding the Company logo.
_applogo	The image used in the About... and splash dialogs holding the product logo.
_pling	The image to go in an error message box.
_info	The image to go in an information message box.
_question	The image to go in a question message box.

Defining Button Windows for Images

To include a button window description in your image specification string, use the following parameters:

{image parameters},{border},{image}

Name	Value
image parameters	The filename, filetype, id, tilesize, and subst parameters, as for a bitmap image specification string.
border (bd)	Specifies the type of border to be drawn round the button.
pushedout (out)	2 pixel pushed-out frame round contents. Suited for decoration buttons. Uses white for top left colour, and BTNSHADOW for bottom right colour. This is the default.
push (p)	Push button borders. Sculpted 3 pixel wide border around contents, displayed either as pushed in or out depending on button select state. Uses standard Windows button colours BTNFACE BTNSHADOW and BTNHILIGHT.
frame (f)	Single pixel frame round contents. Suited for decoration buttons. Frame drawn in Windows WINDOWFRAME colour.
pushedin (in)	2 pixel pushed-in frame round contents. Suited for decoration buttons. Uses BTNSHADOW for top left colour, and white for bottom right colour. This is the default.
shadowed (shad)	1 pixel frame around contents (as for 'frame' style), plus a 3 pixel shadow to the bottom and right, in BTNSHADOW colour.
none (n)	No border.
image (im)	Used to refer to a named button specification for the button contents.

Push button examples

To use an image with label 'Cancel', with push button borders, use:

border=p,type=t, label=Cancel

This is a text push button.

To use the image labelled '_question', with pushed-in border, use:

border=in,image=_question

This is the decorative sculpted question mark you sometimes see in Borland-style dialog boxes.

To use the image labelled '_no', use:

bd=p,im=_no

This is a standard 'no' button used in dialogs.

To frame borders around the image in 'frog.bmp', which is to be stretched to fit the button size, use:

border=frame, file=frog

To display this image, you could use the following AiF escape sequence:

ESC_32 ; 5; 5 ; 15 ; 5 w toad ; border=frame, file=frogESC

This creates a 5x5 image labelled **toad** at (15,5) from the above specification string.

Inbuilt button-images

The following are inbuilt into HostAccess:

Name	Description
_pling	Decorative pushed-in exclamation mark button. Used in message boxes.
_hand	Decorative pushed-in hand button. Used in message boxes.
_question	Decorative pushed-in question mark button. Used in message boxes.
_info	Decorative pushed-in information button. Used in message boxes.
_logo	Decorative pushed-in Company logo button. Used in message boxes.
_applogo	Decorative pushed-in product logo button. Used in message boxes.
_ok	Standard Borland style OK push button.
_cancel	Standard Borland style Cancel push button.
_yes	Standard Borland style Yes push button.

Name	Description
_no	Standard Borland style No push button.
_help	Standard Borland style Help push button.

A

- Accelerator character, 29
- AiF for Windows
 - Common problems, 103
- AiF sequences
 - Clear slots of copied screen regions, 31
 - close DDE link, 112, 211, 214
 - colours, 108
 - detect blinking status, 108
 - detect colour/mono monitor, 108
 - Copying a region of screen, 30
 - cursor off, 110
 - cursor on, 110
 - DDE, 211, 212
 - execute DDE macro, 211, 213
 - field input
 - activate box input, 109
 - activate line input, 108
 - invoke window editor, 109
 - load exit keys, 109
 - host echo off, 109
 - host echo on, 109
 - initiate DDE, 211, 212
 - menus
 - load exit keys, 108
 - Pasting a copied screen region, 31
 - request data from DDE server, 211, 214
 - save environment
 - pop environment, 109

- push environment, 109
 - send data to DDE server, 112, 211, 213
- windows
 - close, 108
 - heading, 108
 - open, 108
- AiF TOOLKIT
 - Escape sequence summary, 12
- AiF Utilities, 106
- Alignment
 - Text, 171

B

- Box Drawing, 160
- Box input, 145
 - Examples, 147
 - Getting a response, 146
- Buttons, 39
 - check boxes, 42
 - Creating a text button, 39
 - example of usage, 44
 - Image, 40
 - radio buttons, 43
 - reading, 45
 - Reading it's check state, 45
 - Reading which is checked, 45
 - setting/clearing, 45
 - Solving problems, 103

C

- CALL
 - Macro, 226
- Capturing screen text, 205

- Cascading menu format, 133
- Centering Text, 171
- Changing Cursor Shape, 166
- Check boxes, 42
- Clear slots
 - Of copied screen regions, 31
- Clipboard, 56
- Closing HOSTACCESS from the host, 204
- Colour
 - Specifying a new, 244
 - Specifying old to be replaced, 245
 - Substitution, 244
- Colours
 - ANSI standard, 113
 - Changing controls, 24
 - Changing default sculpture, 20
 - Intense bit set, 114
 - Resetting to the default, 115
 - Setting default foreground/background, 33
 - Switching ANSI colour mode on/off, 115
- Combo boxes, 51
 - Creating, 51
 - example, 52
 - Hiding and showing, 56
 - Limiting text, 55
 - reading, 53
 - Reading changes, 54
 - Reading current item, 53
 - Reading the contents, 54
 - Reading to see if visible, 54
 - Selecting current item, 55
- Command stack control, 180
- Commands, 82

INDEX

- adding groups, 84
- changing types, 83
- creating, 82
- examples, 85
- reading, 83
- with toolbars, 84
- Control codes, 174
- Control Management -
 - solving problems, 103
- Control response format, 200
- controls
 - root, 32
- Controls
 - accelerator character, 29
 - Alternate message, 28
 - changing colours, 24
 - Creating a group, 28
 - destroying, 23
 - enabling/disabling, 22
 - event reporting, 24
 - groups, 28
 - introduction, 14
 - managing, 22
 - repositioning, 23
 - Return key, 29
 - showing/hiding, 22
 - Using, 28
- Copy
 - A region of screen, 30
- Currency format, 74
- Currency validations, 74
- Cursor
 - Changing, 77

D

- Data extraction, 197
- DDE, 210, 211, 212
 - Client support, 212
 - close link, 211, 214
 - Close link, 214
 - execute macro, 211, 213
 - initiate, 212
 - Initiating a conversation, 212
 - overview, 210
 - poke, 211, 213
 - request data from server, 211, 214

- Requesting data from a server, 214
- Sending commands to the server, 213
- Sending data to a server (Poke), 213
- Server Support, 214
- DDE Sequences, 211
- DDE server support, 215
- DELAY
 - Macro, 226
- DELAYTILL
 - Macro, 226
- delimiters, 15
- Describing Images, 241
- Display Optimisation, 151
- Displaying images, 41
- Displaying Images, 198
- DO
 - Macro, 227
- DOS Gateway, 186
- DOS Integration, 111
- DOS Keyboard Stacker, 188
- Drawing
 - Sculpted lines, 19
- Dynamic Data Exchange, 210
- Dynamic Data Exchange (DDE), 112

E

- Edit boxes, 64
 - Changing the password character, 69
 - Creating, 64
 - Initialising a multi-line edit box, 70
 - manipulating, 68
 - reading, 66
 - Setting selection range, 69
 - Setting the selection range, 55
 - Using the clipboard with, 69
 - validated, 71
- Edit examples, 73
- Emulation
 - Changing terminal type, 207
- END

- Macro, 227
- Environment, 113
- Erase DOS file, 192
- Escape sequences
 - Using, 14
- Escape Sequences, format of, 14
- Event reporting
 - Enabling, 25
- Events, 24
 - Getting, 26
 - Requesting, 26
 - Timed, 93
- Events returned
 - Format, 183
- EXIT
 - Macro, 228
- Exit keys
 - Loading application, 124
 - User response, 124
- Exit Keys, 142

F

- Field input, 140
 - Exit keys, 141
 - Response, 142
 - Types, 140
 - User keys available, 141
- File Transfer, 208
- Focus
 - Setting Input focus, 27
- Fonts
 - Changing, 91
 - Solving problems, 104
 - Using alternate PC, 169
- FOR ... NEXT
 - Macro, 229
- FORMs, 154
 - Examples, 155
 - Files, 154
- Freeze On/Off, 157
- Function keys
 - Programmable, 173

G

- GOTO

- Macro, 229
- ## H
- Host Echo On/Off, 159
- ## I
- IF ... THEN ... ELSEIF
 - Macro, 230
 - Image buttons, 40
 - Images
 - Closing the image
 - application, 199
 - Defining a simple image, 243
 - Defining Button Windows, 250
 - Defining labelled, 247
 - displaying, 41
 - Displaying, 198
 - Displaying multiple, 198
 - Inbuilt, 246
 - Inbuilt button, 251
 - Inbuilt labelled, 249
 - Pre-defining, 242
 - Types, 241
 - INPUT
 - Macro, 231
 - Invoking Windows help, 92
- ## K
- Keyboard Control, 110
 - Keyboard control features, 173
 - Keys
 - Send to Windows
 - applications, 203
- ## L
- LET
 - Macro, 232
 - Line Drawing, 162
 - Line input, 143
 - Getting a response, 144
- ## Lines
- Drawing sculpted, 19
- ## List boxes, 57
- Creating, 57
 - example, 59
 - incremental, 58
 - manipulating, 62
 - reading, 60
 - Setting tabs, 63
- ## M
- Macro, 217
 - Declaring variables, 218
 - Functions, 219
 - Procedures, 222
 - Summary, 223
 - syntax, 217
 - Macros
 - AiF sequence, 172
 - Managing controls, 22
 - Maximise, 200
 - menus
 - commands for, 82
 - Menus, 89
 - activate cascading menus, 108
 - activate pop-down menu, 108
 - AiF, 120
 - AiF options, 121
 - close cascading menus, 108
 - close pop-down menus, 108
 - Colour configuring, 122
 - Configuring selection
 - characters and separators, 123
 - creating, 89
 - Displaying, 89
 - enabling/disabling, 90
 - Enabling/disabling, 90
 - Exit keys, 123
 - load cascading menus, 108
 - load pop-down menus, 108
 - Removing, 90
 - reset cascading menus, 108
 - reset pop-down menus, 108
 - Message boxes
 - modal, 77
 - Minimise, 200
 - Modal message boxes, 77
 - Positioning, 78
 - Returning values to the host, 79
 - Mouse control, 182
- ## P
- Palette
 - Forcing reconstruction, 33
 - Parameter Delimiters, 15
 - PASSKEYS
 - Macro, 232
 - Paste
 - A copied region of screen, 31
 - Pointer
 - Changing, 77
 - Pop-down menus, 125
 - Activating, 127
 - Activating cascading, 132
 - Cascading, 130
 - Clearing, 126
 - Closing, 128
 - Closing pop-down
 - cascading menus, 134
 - Getting a response, 127, 133
 - Loading, 126, 131
 - Resetting, 131
 - Using, 126
 - PRINT
 - Macro, 232
 - Printing to a DOS file/device, 191
 - Problems
 - Buttons, 103
 - Control management, 103
 - Secondary windows, 103
- ## R
- Radio buttons, 43

INDEX

Reading Selected Display
Text, 49
Reading Selected Hidden
Text, 49
Reading String List Size, 49
REM
Macro, 233
Reporting events to the host,
24
Return key
Controls, 29
Root control
creating, 32
miscellaneous functions,
33
reading, 32
Root control features, 32
Root controlmanipulating, 32
Run-time status, 205

S

Save Environment, 150
Scancode keys, 176
List, 176
Switching On/Off, 175
Screen Fill Character, 167
Screen Layout, 16
Screen Modes, 165
Screen sculpting, 17
colours, 17
default colours, 20
example, 20
sculpted boxes, 18
sculpted lines, 19
sculpture mode, 18
Sculpting - solving problems
Problems
Sculpting, 103
Sculpting the screen, 17
example, 20
Secondary windows, 34
activating, 37
creating, 35
destroying, 37
example, 34
hiding/showing, 38
setting output focus, 38
Secondary Windows - solving
problems, 103

SELECT
Macro, 233
Selection boxes, 135
activate, 108
Activating, 137
close, 108
Closing, 138
Getting a response, 138
load, 108
Loading, 135
reset, 108
Resetting, 135
SEND
Macro, 235
SENDTERM
Macro, 235
SENDWIN
Macro, 235
SLOTS, 152
STACK Facility, 153
Special keys
Leader character, 189
Mnemonics, 188
Special output mode, 170
Static labels, 76
Status Bar
changing, 80
hiding/showing, 80
setting pane contents, 80
setting text, 80
String lists, 46
Changing the list to be
displayed, 55
clearing, 50
Creating, 46
examples, 47
Format, 46
manipulating, 47
reading, 49
Reading size, 49
setting special characters,
50
Summary
AiF Utilities, 108
Switching Cursor On/Off,
167
System Message Line, 163
System Message Line Control,
164

T

Terminal echo, 125, 142
Text buttons, 39
Text labels
Specifying, 248
Timed events, 93
Toolbars
creating, 87
Toolbars and toolboxes, 86
adding to, 87
example, 87
hiding/showing, 86
toolboxes
commands for, 82
Toolboxes
Creating, 86
Typeahead Mode, 180

U

Using Control Groups, 28
Utilities, 106

V

Validated edit boxes, 71
Changing the date, 73
creating, 71
currency validations, 74
date validations, 72
integer validations, 71
Special date strings, 72
Verify DOS File or Directory,
196

W

WAIT
Macro, 236
WHILE
Macro, 238
Window Editor, 147
Examples, 149
Getting a response, 148
Windows

AiF sequences, 117
Closing, 118
Control state, 200
Detect if application
running, 202

Headings and footings,
119
Start program, 201
Start program response
format, 201

Windows help
invoking, 92