AlphaWindow
Software Architecture

Revision 1.0

30 May, 1992

Display Industry Association
1007 Elwell Court Suite B
Palo Alto
CA 94303
USA

Tel: 415-967-6888
Fax: 415 960 3522

A Display Industry Association standard implies a concensus of those substantially concerned with its scope and provisions. This standard is intended as a guide to aid the manufacturer, the consumer and the general public. The existence of a Display Industry Association standard does not in any respect preclude anyone, whether he has approved the standard or not, from manufacturing, marketing, purchasing, or using products, processes, or procedures not conforming to the standard. Display Industry Association standards are subject to periodic review and users are cautioned to obtain the latest editions.

CAUTION NOTICE: This Display Industry Association standard may be revised or withdrawn at any time. Purchasers of Display Industry Association standards may receive current information on all standards by calling or writing the Display Industry Association.

Published by:

Display Industry Association
1007 Elwell Court Suite B
Palo Alto
CA 94303
USA

Introduction

This document describes a software architecture layered on top of the existing AlphaWindow Terminal Protocol. It provides a focus for the activities of DIA at this time. The purpose of the architecture is to provide a concrete framework for developers, vendors, system integrators and users to build working systems using components from potentially several different vendors.

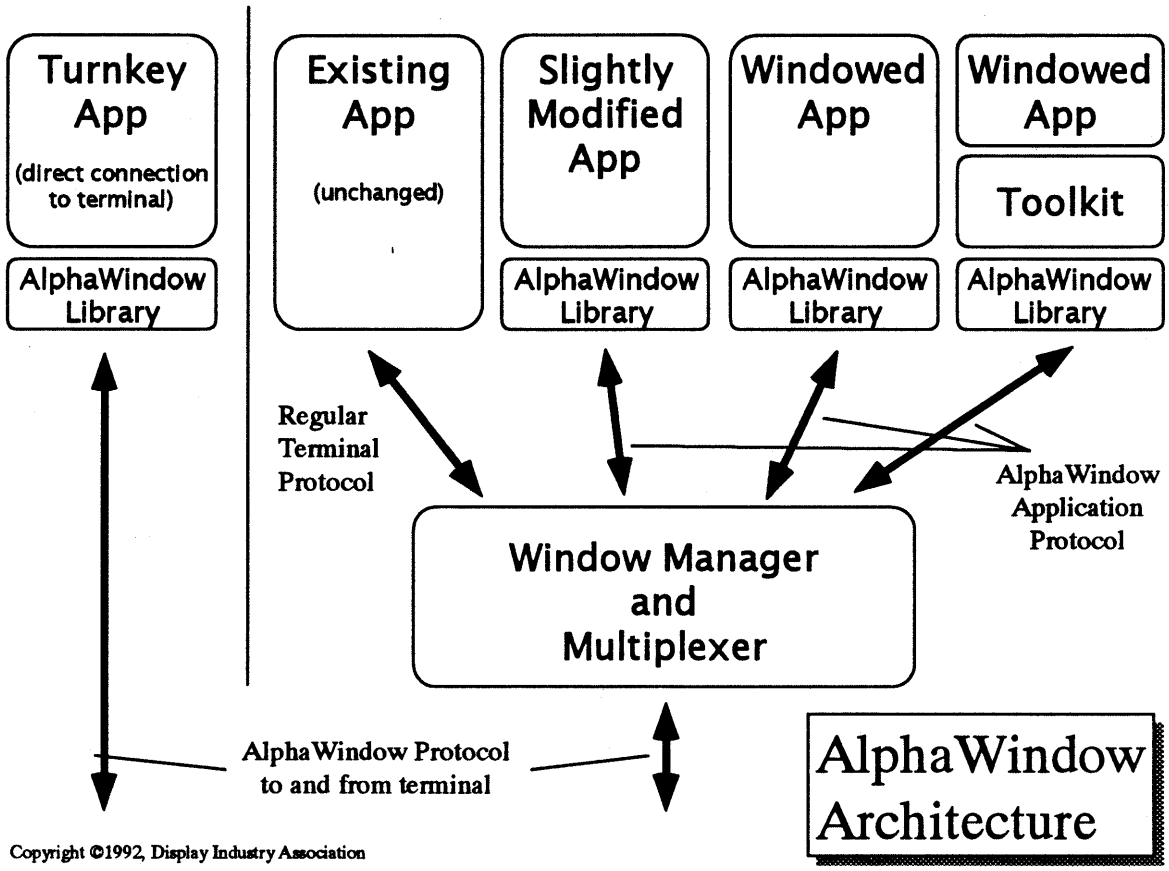
Developers need to be assured that their software will work with the widest possible variety of other components. Vendors want their customers to know that their equipment is compatible with other systems. System integrators need to be able to design and propose systems with confidence that they will function correctly when assembled. All of these reasons will increase user uptake of AlphaWindow technology.

The AlphaWindow protocol is concerned solely with the mechanics of creating and maintaining windows and the use of the mouse. It, and this architecture, do not restrict or define what is drawn or how an application draws its output within these windows. What can be displayed within a window is a function of the underlying terminal emulation.

This architecture specifies four layers of software interface. In ascending order from the physical terminal, these are:

- The AlphaWindow Terminal Protocol. This has been defined by DIA. It is the physical terminal interface and comprises low-level control sequences.
- The AlphaWindow Application Protocol. This is a guide to the use of the Terminal Protocol by applications and window managers. It should be used directly by applications which need low-level control and by higher software layers.
- The AlphaWindow Application Programming Interface Specification. This is a specification for a C language binding to the Application Protocol. It is a higher-level interface and is event-driven in nature.
- Third party toolkits. A number of these already exist and need to be made available in the AlphaWindow environment. This is the level at which applications are already being written for windowing environments such as the X Window System, Microsoft Windows 3 and several "virtual" toolkits.

Architecture Diagram



Notes:

1. The window management and multiplexing functions of the host based software have to be closely coupled. They are shown in the same module within the diagram. This does not imply that the functionality has to be implemented within a single process. It would be possible to have separate window manager and multiplexer processes, but this would require a private protocol between the two to allow the window manager to impose its policy on the display.

2. Window management functionality is not mandatory within a system. Where the same set of clients will always be executed with fixed window sizes and positions, a suitable multiplexer module could be written which gives no window management capability at all. This will only apply to dedicated, special purpose applications.

3. A windowed application may consist of newly written code, or existing code ported to the new environment. DIA expects that existing proprietary toolkits will be ported to the AlphaWindow environment and that other industry standard API's such as OSF/Motif will also be produced by independent software vendors.

Clients

A *client* is an application which the user wants to run on an AlphaWindow terminal. There are four broad categories of software to consider.

1. Existing applications. These will run unchanged with the assistance of the window manager (see below) to create a window for them on the terminal screen. This will be by far the largest use of AlphaWindow terminals, at least initially.

2. Slightly modified applications and toolkits. We envisage that both application developers and toolkit vendors will want to enhance their products to take advantage of the mouse and perhaps resizable windows. It is an aim of this architecture that this level of integration with the terminal should not require large scale software changes. This class of software forms a very large pool of potentially AlphaWindow aware applications.

3. Windowed applications and toolkits. It is very likely that one or more standard GUI's such as OSF/Motif or Windows 3 will be made available for AlphaWindow terminals by third party software vendors. Supporting these new toolkits is an important part of this architecture.

4. Turnkey applications. These are applications which will be written from scratch, or modified, to work directly with the AlphaWindow Terminal Protocol. This is the route by which applications can directly interface to the terminal using the low-level protocol specification. An application which is written in this way is very unlikely to be able to coexist with other applications on the same terminal screen at the same time, so it is not a recommended method of development for general purpose applications.

   AlphaWindow Window Managers are one important class of applications which will be written in this way in order to provide support for multiple concurrent applications.

Window Manager

The term *window manager* is used in the context of this architecture as a generic term to refer to a number of separate, but in practice related areas of functionality:

- Allowing the user to move, resize, restack and otherwise manipulate application windows.

- Multiplexer functionality to support multiple concurrent applications over a single connection.

- Communications protocol handling in order to ensure efficient and reliable host-terminal data transfer.

- Session management in order to allow the user to start and restart applications.

- Support for existing, non-AlphaWindow applications.

The window manager is therefore a key component. Its presence ensures that different applications are able to share the terminal screen peacefully and that the user has a high level of control over the display.

Application Protocol

The application protocol defines the way in which applications should use the AlphaWindow Terminal Protocol to communicate with the terminal or window manager. It also determines the ways in which a window manager is allowed to modify application commands to maintain its window management policy. The protocol is computer and human language independent and has no operating system dependencies. It is layered on top of the character terminal protocol being used, which will obviously depend on the terminal vendor. The Application Protocol is not required to run existing applications, but accommodates the other classes of software which were identified above. It is important that the protocol is clearly and unambiguously defined to ensure that applications from many different vendors can work together on the same terminal. The protocol allows access to all of the application-related GUI functionality of the terminal.

The Application Protocol ensures that different applications will work correctly with window managers from different vendors and that multiple applications may share the same terminal without conflict. A developer may choose to work at this level, coding the appropriate escape sequences into a product, but most companies will find it easier to use the AlphaWindow API or a third party toolkit as described below.

The AlphaWindow API Specification

An AlphaWindow library is a set of functions which provide more or less direct access to the Application Protocol. DIA will define a single reference C language binding called AWlib. Since there are many different languages, application architectures and operating systems with which AlphaWindow terminals will be used, other language bindings may emerge in due course. DIA expects that software vendors will make products available which implement the AWlib specification.

An AlphaWindow library may also be useful to software developers who want only to make minor changes to their software, perhaps to allow the use of the mouse within an application.

Toolkits

A toolkit is a higher level library (possibly layered on top of an AlphaWindow library such as AWlib) which provides GUI functionality in terms of user interface objects such as windows, scrolling lists, menus and so on. DIA anticipates that existing toolkit and development environment vendors such as the many 4GL providers will port their products to the AlphaWindow environment and will actively encourage this process. In addition, DIA should encourage the production of well known graphical API's such as Windows 3.

These toolkits will probably be most useful to developers planning new products or contemplating large scale code changes. Competition in this marketplace will allow developers to choose an API which suits their programmers and their product architecture.

Conclusions

1. Details of the AlphaWindow Application Protocol are contained in a separate document.

2. The AWlib API is defined in the AlphaWindow Application Programming Interface Specification.

3. The Application Protocol and AlphaWindow API definitions will be made available without requirement for membership of DIA.

4. DIA will vigorously promote the AlphaWindow Application Protocol and AWlib specification to ensure its widespread adoption and use.

5. DIA will not attempt to standardise interfaces at higher levels of the architecture. It expects that independent software vendors will migrate existing toolkits and also produce new toolkits, including implementations of standard API's such as OSF/Motif. The marketplace will decide which of these are the most useful.

6. A revenue opportunity is in producing libraries to implement the AWlib specification, toolkits and applications using AWlib, and window managers on top of the existing AlphaWindow Protocol.

7. Although this architecture does not specifically exclude applications which interface directly to the terminal, DIA will clearly explain to application developers the pros and cons of this approach.

AlphaWindow

Terminal Specification

Revision 1.2

April 7, 1992

Display Industry Association

1007 Elwell Court Suite B

Palo Alto, Ca. 94303

Tel: 415 960-1200

or: 415 967-6888

Fax: 415 960-3522

Copyright

All rights reserved

Trademarks

Change Notice

Changes and additions that are incorporated in the present revision are noted with a " | " in the left margin, outside of the text area. Changes will be indicated per paragraph in sections 1 through 4 and per description division in the command section. Please note that these bars will differ each revision and reflect only that a change has been made rather than the exact change. Also note that change bars are not included in the TOC. Each revision is indicated by the date and revision number on the cover page and a revision number on the lower left side of the footer.

Table of Contents

Section 1.
Introduction and Definitions

Introduction

This work is the product of the Technical Committee of the DIA. Much effort has been expended over a 9 month period to produce a specification for a device which has never existed before. The philosophy of the first revision is to develop a specification which is sufficient to produce the first generation of AlphaWindow product without addressing details which would not add significant value. There are known areas of concern in this first release, but on the whole, it is thought to be complete.

Look and Feel Issues

It is important to note that this document does not address the Look-and-Feel issue. This specification defines mechanism not policy. The actual look and feel will be determined by a combination of the host software driving the terminal, and by the terminal firmware's window drawing features.

The window management component is host-based so that full flexibility is achieved in terms of key strokes and mouse interaction utilized by the user to rearrange and restack windows. For a description of the term "Window Manager" - see **Definitions** for the way it is being used here).

Keyboard Issues

The keyboard may be any keyboard the terminal manufacturer chooses in order to fit with a chosen emulation. Enhanced PC style keyboards are desirable as they have developed into the de-facto standard in the industry. It would be useful if at least one special key were provided for use as a windows meta-key. This could then be used in an analogous way to the Alt key in a Microsoft Windows 3.0 environment.

The Terminal Model

The actual terminal emulation implemented is a decision to be made by the individual terminal manufacturers. An ANSI-style terminal such as DEC VT102 or the ANSI.SYS driver for the IBM PC has been used as the primary illustration during development of the specification.

Note that a terminal manufacturer may add value to a windowing terminal as long as the additional value does not conflict with the standard.

Finally, note that this specification does not describe keyboards, emulations, screen appearances, hardware options or toolkit APIs. None of these are within the scope of this document.

Definitions

Various windowing terms used tend to mean different things across different systems.

The following is a list of definitions to help reduce the potential confusion:

Display Server

This is the software component which drives the display. The Display Server is responsible for all window clipping, per-window finite state machine maintenance, etc. It will be implemented in the firmware of the AlphaWindow terminal.

Window Manager

Is responsible for providing the user interface to the windowing system, and for interacting with the user to control and adjust the windowing environment. For flexibility, Window Managers are normally host-based. For example the OSF/Motif Window Manager and the Open Look Window Manager are typical applications which provide their own look and feel.

In the past, Window Manager has meant the software component which actually controls display output to the screen, including window clipping, etc. Nowadays, that component is referred to as the Display Server (See above).

Session

A session is the composite of a host process and a virtual terminal, running over a single, multiplexed connection between the terminal and the host computer. Each session is comprised of a finite state machine with its associated screen memory.

Windowing Terminal Interface

Is the software interface to the terminal itself in order to control the windowing functionality of the terminal from the host. Relative to a character windowing terminal, it is the interface documented in this document.

Window

A window is a viewport onto a virtual terminal screen. More than one window can be mapped onto a specific virtual terminal. Windows may be one of two types; main windows or transparent windows.

A specific main window may be minimized, (ie. in its smallest possible state) or normal (ie. windowed). A minimized window is also referred to as iconic. Maximized windows are implemented as normal windows with the Window Manager recognizing the special condition.

A transient window is identified by a hint provided at creation time. It can be assumed to be of a transient nature in that its existence time is expected to be of short duration and created to support the pull-down menus, dialogue boxes, etc. of a typical windowing interface. The hint is provided so that manufacturers can utilize custom hardware features of their products to implement these types of windows. Manufacturers may choose to implement transient windows as main windows. Although a particular window manager may implement a hierarchical relationship between transients and mains, there is no policy defined here in terms of parent and child relationships. Such relationships should be maintained from the host-based software components.

Main Windows

Typically, individual applications each run in their own individual main window. A main window is very analogous to an OSF/Motif or a Microsoft Windows Main Window.

Transparent Windows

A transparent window is a "see-through" window. A major use of a transparent window is as a bounding box, especially when its border style is set to be a rubber band. Although transparent windows are still related to a specific virtual terminal, no data can ever be visualized within a transparent window.

Virtual Terminal

The equivalent to an ordinary terminal executing an emulation such as VT100. The windowing terminal will support multiple virtual terminals. The exact number varies by vendor and is determined by the resources available to the specific unit. Application programs interact with the virtual Terminal as if it were a physical terminal running only one application.

Window Decorations

There are graphic and text symbols that surround window client areas. Window decorations always appear outside the client area, which is defined by its geometry. In level 1 implementations, the only available window decorations are borders. In level 3, a decoration command is provided which allows a variety of decorations such as caption bar, resizing borders, minimize and maximize icons, etc.

Widgets

There are parts of decorations which have meaning to the window manager and are reported in mouse events. For example, a resizing border consist of eight widgets: top, bottom, left, right and the four corners.

Select

Virtual terminal data can be "selected" by the operator so that it can later be transmitted to the computer. This is usually done by clicking points on the screen with the mouse. After which the host transmits a "select" command, specifying a rectangular area of the virtual terminal. When the terminal makes data "selected", it will highlight the data in some way to show the operator what data has been selected. Later, the selected data can be transmitted to the host computer or de-selected. See AW_SELECT for further explanation.

Section 2.
Power-on Assumptions

Power-on State

The terminal will behave as any ordinary terminal on power-up. A single virtual terminal session will have a single main window associated with it, which will be zoomed to full screen size and have the keyboard attached to it. This ensures that the terminal is immediately usable by any host software such as a login service.

The host system will have some mechanism such as TERMINFO under Unix which defines the base emulation of the product for application software. Each implementation of AlphaWindow terminals will define a set of proprietary characters which are the Control Sequence Introducers for the windowing commands defined in this specification. The codes are chosen by the terminal vendor and will not conflict with the target terminal emulation. A mechanism will be defined on the host to allow applications access to these characters. One example would be an environment variable under Unix. In addition, a keyboard definition file needs to be defined such that Attention Key Sequences can be implemented.

Section 3.
Command Set and Parameter Descriptions

Group Definitions

The following command specification is split-up into a number of functionality groups:

Group 1	Basic Support
Group 2	Mouse support
Group 3	Window decoration
Group 4	Communication

For a terminal to conform with the standards defined in this specification, at least Group 1 functionality must be provided. One or more of the succeeding groups may then be optionally provided in order to supply a more functional terminal.

An initial device enquiry will respond with the actual groups of implemented functionality so that the host software may modify its behavior accordingly.

Group 1 Firmware

Group 1 support is basic windowing. This is the mandatory group of functions that allow the host computer to perform the following functions:

1. Initiate windowing functions, determine terminal resources and windowing capabilities, and enable the other command groups.
2. Control physical terminal screen size.
3. Create and delete virtual terminals.
4. Open, close, position, size, stack, make visible, and form borders and titles for windows into the virtual terminal screens.
5. Route data from and to virtual terminal screens and command processors. Also set keyboard focus such that data typed by the operator is routed to the appropriate sessions.
6. Select virtual terminal screen data for copy and paste operations.
7. Set stacking order for windows.

General Notes

On creation, windows are initially hidden, with all other attributes of indeterminate state so that the host computer may then adjust their geometry and state prior to being revealed.

While hidden, all operations such as geometry and stacking order are still valid.

Group 2 Firmware

Group 2 support is Group 1 support plus the ability to exploit a locally connected serial mouse. (Typically a Microsoft 2-button mouse).

Group 2 support is based around the firmware sending mouse event messages back to the host whenever a mouse event occurs.

In order to avoid flooding the serial connection back to the host, note the following :

The host software may dynamically request the firmware to either deliver, or not deliver, mouse motion events.

Even when mouse motion events have been requested by the host, the firmware should attempt to optimize in order to reduce the actual delivered data. The host software should not really rely on the correctness of the positional information sent within a mouse motion event, it should always make a positional enquiry following a stream of motion events:

1. The host software will usually enable the mouse for solely button reporting until an interaction occurs such that the host would wish to switch motion reporting on. This is for operations of a transient nature such as block-mark operations.
2. Click, double-click and drag are the responsibility of the host software to analyze based on button depression and motion reports sent back to the host. The mouse reports are time-stamped to help the host software to detect double-clicks.
3. The enquiry response is coded separately so that it can easily be spotted by the host software in a potential stream of regular motion events.
4. For Group 3 terminals, the host software may request the MS_WIDGETCROSS mode. In this mode the terminal indicates within the MS_EVENT/MS_STATUS reports an actual widget type that the mouse cursor hotspot crossed at the time of the event.
5. Regardless of the mouse cursor style, the host software is only ever interested in the position of the hotspot. (The host software may specify a particular style, though).
6. The "elapsed time" reported via MS_EVENT/MS_STATUS reports is a "sticky counter". This means that it increments up to a maximum value and then sticks at that point. This is in increments of one-tenth seconds up to a maximum of 10 seconds. It gets reset automatically whenever a MS_EVENT/MS_STATUS report is sent.
7. Whenever the mouse is activated by the host, the terminal should provide a locally driven mouse cursor, by default invisible, to track mouse motion in addition to the text cursor. The mouse cursor moves across the screen un-aided by the window manager. This avoids jerky movements caused by communications delays. When it stops moving long enough, an MS_EVENT message can be sent to the window manager, if appropriate, indicating its new location. Movement is on a character cell basis only.

8. A window can be "attached" to the mouse cursor so that local re-sizing and moving of windows may occur. This operation will typically be performed on a transparent, rubber-band bordered window, but this is not mandatory. The window will move with the mouse cursor, locally.

   There are three kinds of attachment: whole window, single border, and dual border. If the whole window is attached then the whole window simply moves with the mouse. If a single border is attached then the associated dimension can move only horizontally or vertically. If a dual border is attached, two dimensions of the window will expand and contract following mouse movement in any direction (as when a corner is grabbed for resizing).

9. The mouse movements can be limited by the window manager. Mouse movement boundaries can be hard or soft. A hard boundary stops the mouse cursor when encountered and can be used, for example, for confining the mouse cursor to a specific window. Soft boundaries simply result in an MS_STATUS message when encountered. This allows the window manager to highlight areas that the mouse cursor moves through, etc. Only one set of mouse movement hard boundaries and one set of soft boundaries can be set within the terminal.

10. Note that a "mouse" may in fact be any real or emulated device.

Group 3 Firmware

Group 3 support is the ability for the host software to specify which user interface objects, called decorations, should be positioned on window borders.

The type of "decorations" which can be specified to be present are:

Maximize button:	BD_MAX
Minimize button:	BD_MIN
Restore button:	BD_RESTORE
Control menu button:	BD_MENU
Reshape indicators:	BD_SIZE
Horizontal scroll bar:	BD_HSCROLL
Vertical scroll bar:	BD_VSCROLL
Caption bar:	BD_CAPTION

These decorations are typical of the ones utilized by Microsoft Window 3.0, OSF/Motif and Presentation Manager.

If a mouse is in use, then the MS_EVENT and MS_STATUS replies return an indication of whether the mouse cursor hotspot is currently positioned over one of the "widgets" contained in a decoration. All potentially ensuing window manipulation is controlled by the host.

A list of the valid widget identifiers are provided in the MS_EVENT command.

Group 4 Firmware

Group 4 support is for special communications functions. These functions are referred to as a "credit" system which allows for a handshaking alternative to XON/XOFF.

The credit system allows the terminal to start and stop the flow of data for a particular session while allowing data to continue to flow for the other sessions.

Group 4 provides the following functions:

1. It equates a session, which is a virtual terminal within the windowing terminal coupled with an application program running on the host to a "circuit".
2. It indicates to the computer the number of characters of data that it can receive without overflowing its buffer for a particular circuit. The number of characters is expressed in credits, which are actually multiples of characters.
3. It can cancel the credits it has made available.
4. It can inquire into credits available.

General Notes

Group 4 credit handling does not apply to the windowing commands themselves. This includes all groups including Group 4.

Command Structure

Commands are structured in an ANSI style using ascii coded decimal parameters separated by semicolons. A manufacturer specific Windowing Command Introducer character (AW_WCI) is defined for each product. The default is ASCII SOH with a value of 01 Hex. Window manager implementations will be responsible for determining this character sequence for the specific terminals used on a system. The first numeric parameter (P1) determines which AlphaWindow function is being requested. Additional parameters are referred to in this document as P2, P3, etc. A final character, lower case w, is used to indicate the completion of a sequence. Sequences which require text will provide the text after the final character and terminate the text with an ANSI String Terminator (ST). A 7 bit equivalent of ST will also be recognized. Default parameters are indicated by either a "0" or the omission of a parameter.

Once a windowing commnd is initiated, no intervening codes may be imbedded in a command sent from the host to the terminal or from the terminal to the host.

To better illustrate the structure, several sequences are shown below:

AW_WCI 5 ; 2 w	
AW_WCI 5 w	Same as above with P2 Defaulted
AW_WCI ; 2 w	Invalid; P1 is mandatory.

When referring to rows. columns, virtual terminals, or windows, the parameters are based at 1. For instance, the first row n the screen is row 1 and the first column in a row is column 1.

In commands which contain additive values, such as AW_ENABLE_GROUP, the values are added to the set rather than replacing each other. They also are added to the set created by the previous execution of the same command. In these commands, there is always a value (usually the default) which clears everything. A typical AW_ENABLE_GROUP will first clear to group 1 only, then additively include 2,3, or 4.

To better illustrate the structure, several sequences are shown below:

AW_WCI 33 ; 1 ; 2 w	Set groups enabled to 1 and 2
AW_WCI 33 ; 3 w	Add group 3 to any previously enabled groups
AW_WCI 33 1 w	Reset to group 1 only

The symbol "ellipse" (;...;) means the values in the command are additive and that any number of values can be present

Common Parameter Definitions

Parameters which are used in many sequences are defined below:

W_Handle	A Window Handle. This handle is a unique value defined by the terminal at window creation time and is used to uniquely identify windows.
Virtual_Width	The number of columns in a virtual terminal screen. This value is specified at creation of the Virtual Terminal, but may change in the course of an emulation when applications affect the size of the virtual terminal display. An example is an application which switches between 80 and 132 column mode.
Virtual_Height	The number of rows in a virtual terminal screen. This value is specified at creation of the Virtual Terminal, but may change in the course of an emulation when applications affect the size of the virtual terminal display. An example is an application which switches between 24 and 42 row mode in a Wyse 60 emulation.
VT_Handle	A Virtual Terminal Handle. This is a unique value defined by the terminal at virtual terminal creation time to uniquely identify a virtual session.

Section 4.

Communications Protocol

Introduction

The communications protocol of this specification is a level of functionality that provides session switching, session specific flow control, and session specific break events. Host computers already have several standard protocols to manage flow control with existing terminals. XON/XOFF, XPC, and DTR/DSR are the most common to the ASCII/ANSI terminal market and are almost universally supported. Multiple sessions on a single device introduces some unique problems which are often not covered with existing host to terminal protocols. These problems are further aggravated by the use of terminal servers in network installations and their inherent latency.

This section describes the concepts which are supported by all AlphaWindow terminals and the commands which are provided in Group 4 which may or may not be supported by individual hosts or terminals.

Special Characters

Three special characters are defined and supported by all AlphaWindow terminals:

AW_WCI

AW_MPI

AW_LITERAL

These characters, or character sequences, are defined in a profile on the host for each manufacturers product:

AW_WCI — The Windowing Command Introducer is a special character which indicates an AlphaWindow Command. The character is chosen by the vendor such that it has no meaning in the base terminal emulation, and therefore can always be assumed to indicate communication with the Display Manager.

The default value is ASCII SOH (Hex 01)

AW_MPI — The Multiplexor Introducer is a special character which indicates a routing command. The character is chosen by the vendor such that it has no meaning in the base terminal emulation. The routing command can be sent either from terminal to host or from host to terminal. Its purpose is to route display data or keyboard data to their appropriate sessions. The connection between a virtual terminal and a program is sometimes referred to as a circuit. The ID of a circuit is its virtual terminal handle. AW_MPI need not be sent prior to each character, only when routing in either direction changes. For performance purposes, an AW_MPI command has a special format:

AW_MPI <HANDLE_PACKED>

The "packed" virtual terminal handle is packed by adding the binary value of the virtual terminal to 30 hex, producing a single ASCII character.

The default value is ASCII STX (Hex 02)

AW_LITERAL This character is used to indicate that the next character received in the data stream should not be interpreted, but should be passed through as data to the currently selected session. This is necessary to allow the AW_WCI and AW_MPI characters to be embedded in data or passed through the terminal to local printers. AW_LITERAL is indivisible from the character paired with it. There can be no intervining characters between AW_LITERAL and the code to be passed through.

The default value is ASCII DLE (Hex 10)

AW_XON, AW_XOFF DC1 and DC3 characters in the data stream are replaced by AW_XON and AW_XOFF respectively. This is to prevent the device drivers from doing improper flow control interpretation.

AW_XON default value is ASCII DC2 (Hex 12)

AW_XOFF default value is ASCII DC4 (Hex 14)

AW_BREAK The host software, when receiving AW_BREAK, will interpret it as a break condition and can substitute it for a real break condition.

AW_BREAK default value is ASCII EOT (Hex 04)

Power-on State

Initially AlphaWindow terminals are single session terminals as defined in Section 2 under Initial State. Only AW_BEGIN commands will be recognized. AW_LITERAL, AW_MPI, AW_XOFF, and AW_XON are all ignored until AW_BEGIN is received by the terminal.

Windowing Commands

Introduction

This section contains the presently defined commands. Below is a list of the commands separated by groups:

Group 1	Px	Group 1 *Cont.*	Px	Group 2	Px
AW_ATTENTION	1	AW_RDISPSZ	61	MS_ATTACH	201
AW_BEGIN	7	AW_RESTORE	62	MS_BOUND	205
AW_CLOSE_WIN	9	AW_REXIT	63	MS_ENQ	209
AW_CREATE_VT	13	AW_REMUL	64	MS_EVENT	213
AW_DA	17	AW_RGEOM	65	MS_GCONFIG	217
AW_DATA	21	AW_RVT	73	MS_MODE	221
AW_DELETE_VT	25	AW_RWIN	77	MS_MOVE	225
AW_DESELECT	29	AW_SBORDER	81	MS_RCONFIG	229
AW_ENABLE_GROUP	33	AW_SDISPSZ	85	| MS_SHORT_EVENT	2
AW_EXIT	37	AW_SELECT	89	MS_STYLE	233
AW_FREEZE	38	AW_SEND	91		
AW_GBORDER	39	AW_SETATTN	93	**Group 3**	**Px**
AW_GDISPSZ	41	AW_SGEOM	97	AW_SDECORATION	261
AW_GEMUL	43	AW_SKBD	101		
AW_GGEOM	45	AW_STACK	105	**Group 4**	**Px**
AW_OPEN_WIN	53	AW_TITLE	109	| AW_ADDCREDIT	3
AW_RATTN	54	AW_TITL_HILIT	111	AW_CREDITENQ	309
AW_RBEGIN	55	AW_TRACK	113	AW_RCREDIT	313
AW_RBORDER	57	AW_VISIBILITY	117	AW_ZEROCREDIT	317
AW_RDA	59				

Numbering and command arrangement

P1=X numbering is arranged in steps of 5 or X + 4 = the next command number. This was done to allow new commands to be inserted in between to maintain an alphabetical form for as long as practical with certain assumptions made as to the necessary expansion of each group. Also, an amount of "dead space" was allowed between each group. Group 1 starts with series 1 through 200, group 2 starts with 201 through 260, group 3 starts at 261 through 300, group 4 starts at 301 and ends at 400. Because numbering is arbitrary, 401 and on can be any group. When adding new commands - fill the presently designated areas first.

| In addition, certain commands have been shortened to improve performance. Specifically MS_SHORT_EVENT has been implemented with Px=2,and AW_ADDCREDIT has been implemented with Px=3.

AW_ATTENTION $P_1 = 1$

Group: 1

Sequence: AW_WCI 1 ; <Key_Idn> w

Description: Sent to the host whenever a set of keys predefined by a AW_SETATTN is struck. AW_ATTENTION is transmitted to the host when the key combination is made (held down simultaneously). Any subset key chords of an attention are also sent.

The order of subsets sent to the host is at the discretion of the terminal vendor.

The timing required to detect a key combination is at the discretion of the terminal vendor.

Reply: [None]

Direction: Host ⟵ Terminal

AW_BEGIN P_1= 7

Group: 1

Sequence: AW_WCI 7 w

Description: Request from the host to the terminal to begin using the AlphaWindow protocol. If confirmation is not received within a specified time, the host will assume the terminal is not an AlphaWindow terminal. All other AW_WCI commands are ignored until AW_BEGIN is received by the terminal. The terminal will not recognize special characters or AlphaWindow commands until AW_BEGIN is received.

An AW_BEGIN received while the terminal is in AlphaWindow mode will place the terminal in the same state as AW_BEGIN received after power-on (see power-on state).

Reply: [AW_RBEGIN]

Direction: Host ⟶ Terminal

Note: All virtual terminals, windows, attention keys, etc., are cleared and only wallpaper shows on the display.

AW_CLOSE_WIN P_1= 9

Group: 1

Sequence: AW_WCI 9 ; <W_Handle> w

Description: Close window. Window is deleted and resources freed. W_Handle is mandatory, if missing or invalid, the command is ignored.

If the keyboard focus is assigned to the window being closed, the focus becomes unassigned.

Reply: [None]

Direction: Host ⟶ Terminal

Note: Does not affect associated virtual terminal session.

AW_CREATE_VT P_1= 13

Group: 1

Sequence: AW_WCI 13 ; <Virtual_Width>; <Virtual_Height>; <Maximum_Width>; <Maximum_Height>; <Private_Hint> w <Name>ST

Description: Creates a virtual terminal session, including a finite state machine, with virtual terminal size specified by Virtual_Width, Virtual_Height. Virtual_Width and Virtual_Height default to the normal height of virtual terminal screen being created. Virtual_Width and Virtual_Height are the sizes of virtual terminal screens.

Maximum_Width and Maximum_Height indicate the maximum sizes that the virtual terminal may become due to application screen size changes, at which time an unsolicited AW_RVT will be sent to the host. The default is Virtual_Width and Virtual_Height.

Name is from AW_REMUL; default is the first emulation reported in the AW_REMUL list.

Private_Hint indicates that the virtual terminal will be used by the window manager. The terminal may use this hint to implement vendor specific features.

P_6	<Private_Hint>
1	PH_NORMAL
2	PH_PRIVATE

Reply: [AW_RVT]

Direction: Host ⟶ Terminal

Note: If the terminal is unable to create a virtual terminal with the specified virtual width and height, the terminal can create virtual terminal with a width and height of its own choosing and report this via AW_RVT.

Virtual_Width and Virtual_Height are unconnected with P_Height and P_Width as specified in AW_SDISPSZ or AW_RDISPSZ, or with any application size commands.

Group: 1

Sequence: AW_WCI 17 w

Description: Windowing terminal device attribute query.

Reply: [AW_RDA]

Direction: Host ⟶ Terminal

AW_DATA P_1= 21

Group: 1

Sequence: AW_WCI 21 w <Data> ST

Description: Lines are separated with carriage return (CR) characters. No carrage return is transmitted after the last line. The actual keyboard should be locked during this transmission. The data immediately follows the final character of the sequence. A string terminator character is sent upon completion.

Reply: [None]

Direction: Host ⟵ Terminal

Note: This is to support cut and paste. Data is selected by AW_SELECT.

AW_DELETE_VT P_1= 25

Group: 1

Sequence: AW_WCI 25 ; <VT_Handle> w

Description: Deletes a virtual terminal session. Deleting a VT will automatically close any open windows associated with that VT. If keyboard focus is on a window of the virtual terminal, only attention keys will be transmitted after this command is executed. VT_Handle is required and the command will be ignored if it is omitted or invalid.

Reply: [None]

Direction: Host ⟶ Terminal

AW_DESELECT $P_1 = 29$

Group: 1

Sequence: AW_WCI 29 w

Description: Un-selects the virtual terminal screen data selected by the previous AW_SELECT. The data is then un-highlighted.

Reply: [None]

Direction: Host ⟶ Terminal

See Also: AW_SELECT

AW_ENABLE_GROUP $P_1 = 33$

Group: 1

Sequence: AW_WCI 33 ; <Group> ;...; <Group> w

Description: Enables respective alpha window command groups. Group 1 is automatically included in all selections. Groups are additive.

P_n	Group
1	**Group 1** ***ONLY***
2	Group 2 *and* 1
3	Group 3 *and* 1
4	Group 4 *and* 1

Reply: [None]

Direction: Host ⟶ Terminal

Note: Additional groups are reserved.

AW_EXIT P_1= 37

Group: 1

Sequence: AW_WCI 37 w

Description: Request to exit AlphaWindow mode. The terminal returns to it's power-up state. Should a power failure occur, the operator should press a special key combination, causing the terminal to send an AW_EXIT to the host computer, at which time the computer can exit windowing gracefully.

Reply: [AW_REXIT]

Direction: Host ⟷ Terminal

Note: Even when the terminal is not in AlphaWindow mode, it should be able to send AW_EXIT.

AW_FREEZE_REF $P_1 = 38$

Group: 1

Sequence: AW_WCI 38 ; <Freeze> w

Description: Freeze or Unfreeze screen refreshes. This command allows the computer to prevent the display of incomplete results of resizing, stacking, visibility, etc. The actual appearance of the screen when this command is used is at the discretion of the vendor.

P_2	<Freeze>
1	**Unfreeze**
2	Freeze

When Freeze is used, refreshes of the display are delayed untill Unfreeze is used.

Reply: [None]

Direction: Host ⟶ Terminal

AW_GBORDER $P_1 = 39$

Group: 1

Sequence: AW_WCI 39 ; <W_Handle> w

Description: Request the terminal to return an AW_RBORDER, indicating the size of the border segments.

Reply: [AW_RBORDER]

Direction: Host ⟶ Terminal

AW_GDISPSZ P_1= 41

Group: 1

Sequence: AW_WCI 41 w

Description: Get display size palette (Refers to the physical screen size).

Reply: [AW_RDISPSZ]

Direction: Host ⟶ Terminal

Note: Screen size is vendor specified.

Group: 1

Sequence: AW_WCI 43 w

Description: Get emulation support summary.

Reply: [AW_REMUL]

Direction: Host ⟶ Terminal

AW_GGEOM $P_1 = 45$

Group: 1

Sequence: AW_WCI 45 ; <W_Handle> w

Description: Get geometry of specified window. W_Handle is required. If invalid or missing, AW_RGEOM with a window handle of zero is returned.

Reply: [AW_RGEOM]

Direction: Host ⟶ Terminal

AW_OPEN_WIN $P_1 = 53$

Group: 1

Sequence: AW_WCI 53 ; <VT_Handle> ; <Window_Type> ; <Transient_Flag> w

Description: Open window onto specified virtual terminal. Window may be of type main, or transparent. Window is initially hidden with all other attributes of indeterminate value. Transient flag indicates whether the window is expected to be of a temporary nature. Note that the Transient flag is simply a hint to the terminal that this window is expected to be in existence for a short time. The terminal may choose to ignore the hint if it so desires, or it may choose to perform some degree of optimization for transients, if relevant. If VT_Handle is missing or invalid or if the terminal fails to open the window due to a lack of memory or some other failure, the reply AW_RWIN will return a window handle of zero.

P_3	<Window_Type>
1	WT_MAIN
2	WT_TRANSPARENT

P_4	<Transient_Flag>
1	TF_NORMAL
2	TF_TRANSIENT

Reply: [AW_RWIN]

Direction: Host ⟶ Terminal

AW_RATTN $P_1 = 54$

Group: 1

Sequence: AW_WCI 54 ; <Key_ID>; <Status> w

Description: Indicates the success or failure of an AW_SETATTN command.

P_3	Status
1	**Success**
2	Failure

Reply: [None]

Direction: Host ⟵ Terminal

AW_RBEGIN $P_1 = 55$

Group: 1

Sequence: AW_WCI 55 w

Description: Returned to the host upon receipt of an AW_BEGIN sequence acknowledging usage of the alphaWindow commands.

Reply: [None]

Direction: Host ⟵ Terminal

AW_RBORDER $P_1 = 57$

Group: 1

Sequence: AW_WCI 57 ; <W_Handle>; <Bd_Sz_Top>; <Bd_Sz_Rt>; <Bd_Sz_Bot>; <Bd_Sz_Lt> w

Description: Response to AW_GBORDER. Top and bottom values indicate the number of character rows used to create the border, including decorations. Left and right border thicknesses indicate the number of character columns used to create the border. This allows the window manager to determine total window size on the display for calculating new positions during tiling and cascade arrangements. If Bd_Sz_Top, Bd_Sz_Rt, Bd_Sz_Bot, or Bd_Sz_Lt are omitted, their values default to 1.

Reply: [None]

Direction: Host ⟵ Terminal

Note: Window borders, including decorations, sit outside the window client area.

AW_RDA $P_1 = 59$

Group: 1

Sequence: AW_WCI 59 ; <Major_Rev>; <Minor_Rev>; <Group>;...;<Group> w

Description: Sent in response to AW_DA. Specifies which functionality groups the firmware supports. <Group>;...;<Group> is an explicit list of groups supported. If groups 1, 2, 3 and 4 are supported, then four parameters will be returned in addition to the first three parameters indicating an AW_GROUPS sequence. Group 1 support is mandatory. Major_Rev is the digit to the left of the decimal point and Minor_Rev is the digit to the right of the decimal point reflecting the revision found on the cover page of this specification.

Reply: [None]

Direction: Host ⟵ Terminal

AW_RDISPSZ P_1= 61

Group: 1

Sequence: AW_WCI 61 ; <Width_I>; <Height_I>; <Width_C>; <Height_C>; <Width_Lo>; <Width_Hi>; <Height_Lo>; <Height_Hi>; <Width>; <Height> ; <Width>; <Height>;..; <Width>; <Height> w

Description: Report display size palette. Icon widths and height, current display size, all possible display size pairs. AW_RDISPSZ is sent in response to AW_GDISPSZ.

Width_I and Height_I are the number of character columns and rows the terminal uses to produce ICONS (figures which depict minimized windows). This is the total width and height including borders and titles.

Width_C and Height_C are the number of columns and rows on the current physical display.

Width_Lo and Width_Hi specify a range within which the host can specify, using AW_SDISPSZ, any display width (for those terminals capable).

Height_Lo and Height_Hi specify a range within which the host can specify any display height.

Width and Height are repeated pairs indicating all possible width and height specific combinations of display size that the terminal allows.

Reply: [None]

Direction: Host ⟵ Terminal

Note: Response is based on resources available at the time AW_GDISPSZ was received.

AW_RESTORE $P_1 = 62$

Group: 1

Sequence: AW_WCI 62 w

Description: Sequence from terminal to host indicating that the terminal's environment has been corrupted, such as a power failure. This sequence can be sent at any time, including power-up. The host computer can then re-create the necessary virtual terminals and windows to restore the terminal's state. Typically, this is accomplished by some special key combination.

Reply: [None]

Direction: Host ⟵ Terminal

AW_REXIT $P_1 = 63$

Group: 1

Sequence: AW_WCI 63 w

Description: Confirmation that AlphaWindowing has ceased. All virtual terminals are deleted and all windows closed. The terminal reverts to the power-on default condition.

Reply: [None]

Direction: Host ⟷ Terminal

AW_REMUL P_1= 64

Group: 1

Sequence: AW_WCI 64 w <name>;...; <name> ST

Description: Report emulations supported by the terminal. Names are separated by semicolons. List is terminated by ST.

Reply: [None]

Direction: Host ⟵ Terminal

AW_RGEOM P_1= 65

Group: 1

Sequence: AW_WCI 65 ; <W_Handle>; <W_State>; <X>; <Y>; <Width>; <Height>; <Virt_X>; <Virt_Y>; <Virtual_Screen Width>; <Virtual_Screen Height>; <P_Width>; <P_Height>; <Caption_Width> w

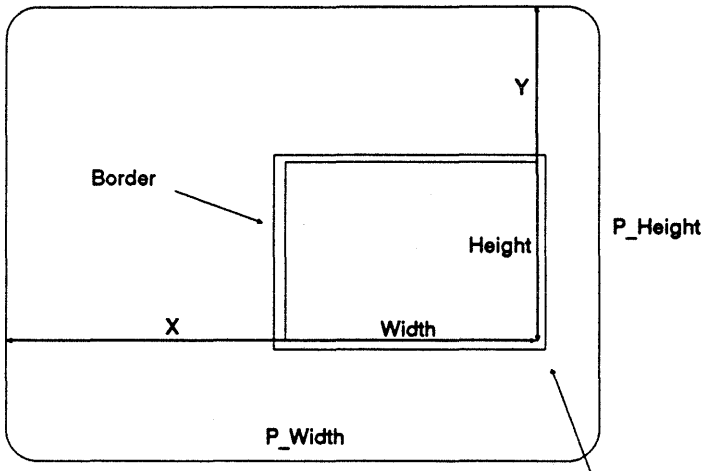
Description: Report geometry of specified window. X, Y, Width, Height are relative to physical terminal screen. Virt_X, Virt_Y are the offset into the virtual screen. Virtual_Width, Virtual_Height are the virtual screen size. W_State is WS_MIN or WS_NORM for minimized or non-minimized states respectively. AW_RGEOM is sent in reply to AW_GGEOM. Caption width returns the current maximum caption bar title text size. When state is WS_MIN, X and Y, refer to the Icon location. In all cases, X and Y refer to the bottom right corner of the window, relative to the top left corner of the physical terminal display.

P_3	<W_State>
1	WS_NORM
2	WS_MIN

Reply: [None]

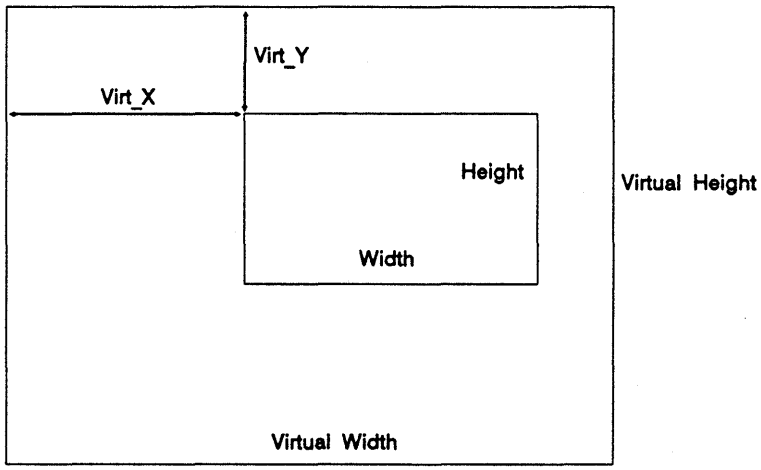
Direction: Host ⟵ Terminal

Note: See following page illustrations of physical and virtual screen/display areas.



Physical Terminal Display

Note: The border, including decorations, sit outside of the window boundries.



Virtual Session Memory Image

AW_RVT P_1= 73

Group: 1

Sequence: AW_WCI 73 ; <VT_Handle>; <Virtual_Width>; <Virtual_Height> w

Description: Sent in response to an AW_CREATE_VT request. VT_Handle is assigned by the terminal. On fail, for whatever reason, a VT_Handle of zero is returned. This sequence can be unsolicited as a result of an application's display size change sequence. Virtual_Width and Virtual_Height are virtual terminal screen height and width. The range of VT_Handle is 1 to 4F hex to allow for packed VT_Handle parameters in AW_MPI sequences.

Reply: [None]

Direction: Host ⟵ Terminal

AW_RWIN $P_1 = 77$

Group: 1

Sequence: AW_WCI 77 ; <W_Handle> w

Description: Sent in response to an AW_OPEN_WIN request. Window handle is assigned by the terminal. On fail, for whatever reason, a window handle of zero is returned.

Reply: [None]

Direction: Host ⟵ Terminal

AW_SBORDER P_1= 81

Group: 1

Sequence: AW_WCI 81 ; <W_Handle>; <Border_Style> w

Description: Set border style for specified window.

P_3	<Border_Style>
1	BS_THICKNORMAL
2	BS_THIN
3	BS_NONE
4	BS_THICKBOLD
5	BS_GHOSTOUTLINE

Reply: [None]

Direction: Host ⟶ Terminal

See Also: AW_SDECORATION

Note: If border style is BS_NONE, the window decorations are still displayed.

AW_SDISPSZ $P_1 = 85$

Group: 1

Sequence: AW_WCI 85 ; <P_Width>; <P_Height> w

Description: Set physical screen size. P_Width, P_Height must be a valid pair reported in AW_RDISPSZ. The terminal will refresh the terminal screen. Invalid parameters causes the sequence to be ignored. It is the window manager's responsibility to ensure that window dimensions, mouse boundaries, etc., remain within proper range.

Reply: [None]

Direction: Host ⟶ Terminal

See Also: AW_GDISPSZ, AW_RDISPSZ

AW_SELECT P_1= 89

Group: 1

Sequence: AW_WCI 89 ; <VT_Handle>; <Start_Row>; <Start_Col>; <End_Row>; <End_Col>; <Select_Mode> w

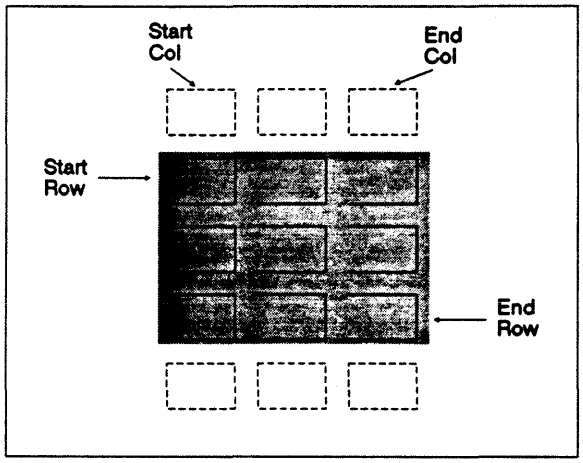
Description: The specified rectangle of the specified virtual terminal screen is "selected" (used by AW_SEND) and is highlighted by the terminal. Any previously selected data is de-selected by the terminal.

Coordinates are relative to the virtual terminal screen. Select mode indicates ether rectangular or line-wrapped highlighted area the actual method of highlighting is at the prerogative of the terminal manufacturer (Eg. reverse video block). Wrap occurs at the end of virtual lines. and wraps to the beginning of subsequent lines. Once data is selected, it can scroll or move and the selection will move with it. Changes to selected data and their results are undefined for this specification. Only one selection can be active in the Alphawindow terminal at a time.

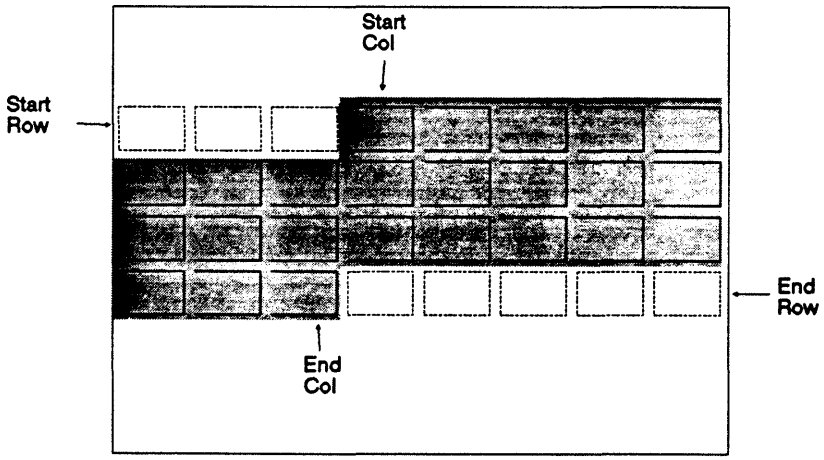
P_7	<Select_Mode>
1	HS_RECT
2	HS_WRAP

Reply: [None]

Direction: Host ⟶ Terminal



Virtual Terminal Display, Select_Mode = HS_RECT



Virtual Terminal Screen, Select_Mode = HS_WRAP

AW_SEND P_1= 91

Group: 1

Sequence: AW_WCI 91 w

Description: Request the currently selected data to be sent to the host. If there is no data selected, then AW_DATA will be returned with no data.

Reply: [AW_DATA]

Direction: Host ⟶ Terminal

AW_SETATTN $P_1 = 93$

Group: 1

Sequence: AW_WCI 93 ; <Key_Idn>; <Key_Number>;..;<Key_Number> w

Description: Define attention keys. Mechanism for host to define to terminal a range of key sequences and identifiers to return.

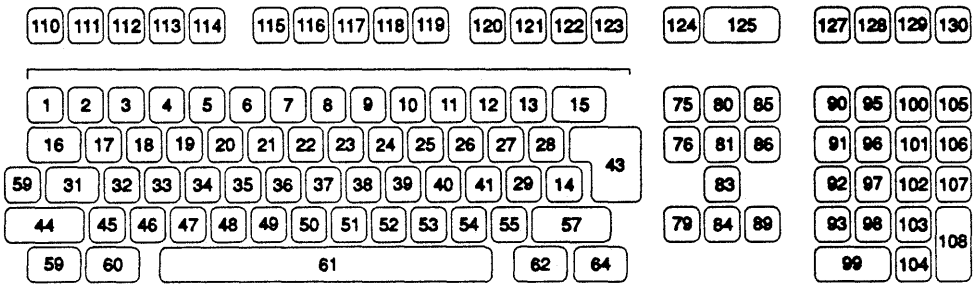
Key numbers are assigned to keyboard keys according to IBM EPC and DEC VT420 keyboard standards. In addition, individual terminal manufactures can provide additional documented key codes for their own keyboards. Key numbers are in ASCII decimal format. All Key numbers in a single AW_SETATTN command form a "chord of keys" which when struck at the same time, will cause the terminal to transmit an AW_ATTENTION sequence to the host with the KEY_IDN code specified. AW_SETATTN commands are cumulative. If KEY_IDN is absent, all AW_SETATTN are cleared. If Key_Number is absent, SETATIN for KEY_IDN is cleared.

EPC (enhanced 101/102 keys)and VT420 keyboards are shown on the following page. Corresponding decimal key numbers are also shown on the following page.

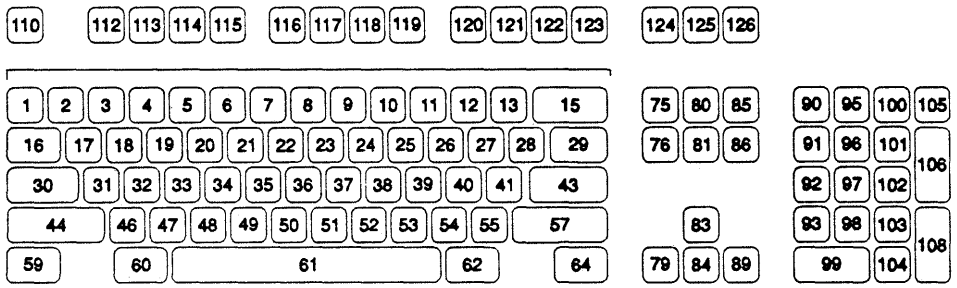
Once a key chord has been set using this command, it will no longer be available as session data.

Reply: [AW_RATTN]

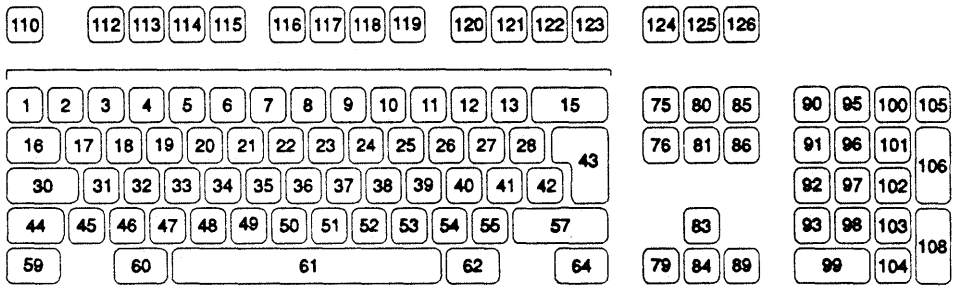
Direction: Host ⟶ Terminal



VT320 Style Keyboard



AT Style Keyboard



AT Style Keyboard (International)

AW_SGEOM $P_1 = 97$

Group: 1

Sequence: AW_WCI 97 ; <W_Handle>; <W_State>; <X>; <Y>; <Width>; <Height>; <Virt_X>; <Virt_Y> w

Description: Set geometry of specified window. X, Y, Width, Height are relative to physical terminal screen. Virt_X, Virt_Y are relative to the virtual screen. State is WS_MIN or WS_NORM for minimized or non-minimized W_State(s) respectively; when the state is WS_MIN, X and Y are relative to the icon location.

X and Y refer to the row and column of the bottom-right hand corner of the window relative to the top left corner of the physical terminal display. This allows the window to be positioned off the screen using positive integer parameters.

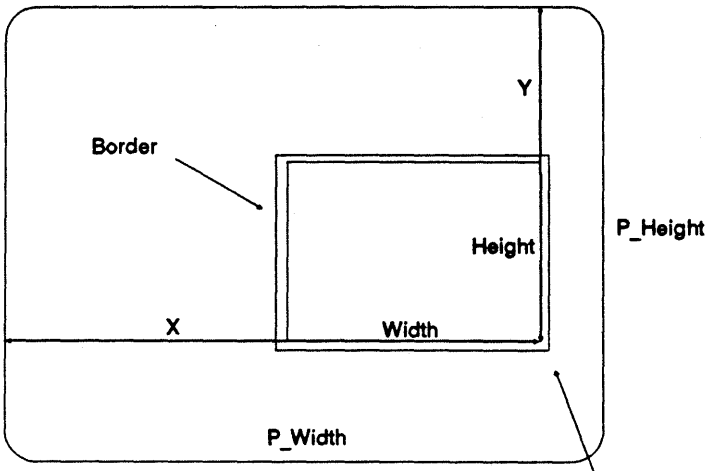
P_3	<W_State>
1	WS_NORM
2	WS_MIN

Reply: [None]

Direction: Host ⟶ Terminal

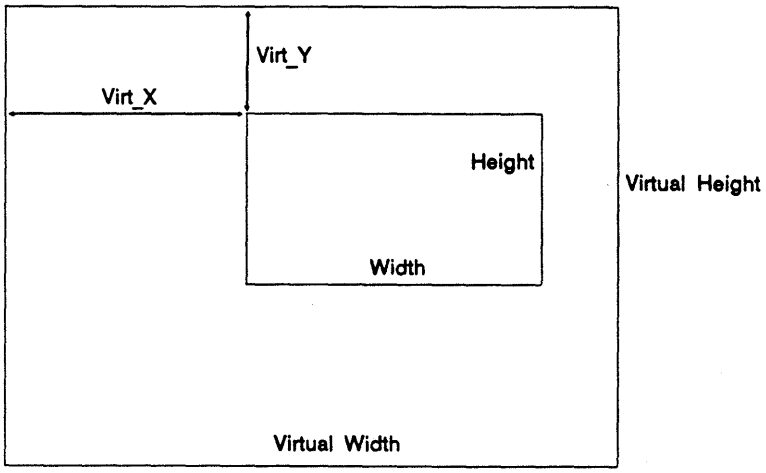
Example: AW_WCI 97 ; 1 ; ; 1 ; 1 ; 80 ; 24 ; 1 ; 1 w

Explanation: Open a window which occupies an entire 80 x 24 physical display and displays an entire 80 x 24 virtual terminal.



Physical Terminal Display

Note: The border, including decorations, sits outside of the window boundries.



Virtual Session Memory Image

AW_SKBD P_1=101

Group: 1

Sequence: AW_WCI 101 ; <W_Handle> w

Description: Assign keyboard focus to specified window. Active cursor appears in the specified window and terminal sends a AW_MPI routing command with the virtual terminal associated with the window each time the keyboard data is sent to the host (if different from the previous virtual terminal). This is completely separate from host display data which is routed from the host to a virtual terminal via AW_MPI. If W_Handle is invalid, the command is ignored.

Reply: [None]

Direction: Host ⟶ Terminal

Note: When keyboard focus is unassigned, only attention keys can be transmitted and no active cursor appears.

AW_STACK P_1=105

Group: 1

Sequence: AW_WCI 105 ; <W_Handle>; <Stack_Request> w

Description: Promote or demote specified window in window stack. Promotion is always to top most, demotion is always to bottom most.

P_3	<Stack_Request>
1	**SR_PROMOTE**
2	SR_DEMOTE

Reply: [None]

Direction: Host ⟶ Terminal

AW_TITLE P_1=109

Group: 1

Sequence: AW_WCI 109 ; <VT_Handle> ; <W_Handle>; <Virt_Row> ; <Virt_Col> ; <Length> w

Description: Set window title. The text located in the virtual terminal screen specified by VT_Handle at the location specified by <Virt_Row> and <Virt_Col> for a length specified by <Length> becomes the title for the window specified by <W_Handle>. After execution of AW_TITLE, the text can be deleted from the virtual terminal screen. If length = 0, no title will be displayed.

Reply: [None]

Direction: Host ⟶ Terminal

Note: Null text erases the title

See Also: AW_GEOM

AW_TITL_HILIT P_1=111

Group: 1

Sequence: AW_WCI 111 ; <W_Handle>; <Highlight_Request> w

Description: Highlight or Un-highlight a window's title according to Highlight_Request

P_3	Highlight_Request
1	**HR_NORMAL**
2	HR_HILITE

Reply: [None]

Direction: Host ⟶ Terminal

AW_TRACK P_1=113

Group: 1

Sequence: AW_WCI 113 ; <W_Handle>; <Track>;...;<Track> w

Description: Sets or resets cursor tracking on specified window. If cursor tracking is set on, the terminal should ensure that the current cursor position is visible by scrolling the virtual screen relative to the window. Tracking should occur regardless of the activitie which requires the cursor to be scrolled into view, i.e., resize and move. Horizontal and vertical tracking may be set independently. Tracking selections are cumulative.

P_3	<Track>
1	CS_NONE
2	CS_HTRACK
3	CS_VTRACK

Reply: [None]

Direction: Host ⟶ Terminal

Note: Tracking occurs when cursor is outside of boundary, if boundary is moved away from the cursor - cursor tracking must start.

AW_VISIBILITY P_1=117

Group: 1

Sequence: AW_WCI 117 ; <W_Handle>; <Visibility> w

Description: Hide or reveal specific window, (client area, borders, decorations, and minimized window icorns), or all windows if Window handle is zero. It is the window manager's responsibility to ensure that the window's geometry and decorations are appropriate prior to using this command.

P_3	<Visibility>
1	**VR_REVEAL**
2	VR_HIDE

Reply: [None]

Direction: Host ⟶ Terminal

Note: When VR_HIDE is used, decorations and borders are hidden.

MS_ATTACH P_1=201

Group: 2

Sequence: AW_WCI 201 ; <W_Handle>; <X>;<Y>; <Attachment_Type> w

Description: Attaches the specified window to the mouse at the specified attachment anchor point (X,Y). If the mouse position is different from the anchor point at the time this command is received , the difference is used to adjust the geometry of the attached window. Any type of window, including icons, can be attached to the mouse. The esthetics quality of the results is the responsibility of the manufacturer. Attachment type BD_MOVE_ALL causes adjustments of the window position with mouse movement and all other attachment types cause adjustments to window size. Only one MS_ATTACH is allowed at a time.

P_4	<Attachment_Type>
1	**BD_DETATCH**
2	BD_STRETCH_N
3	BD_STRETCH_E
4	BD_STRETCH_S
5	BD_STRETCH_W
6	BD_STRETCH_NE
7	BD_STRETCH_SE
8	BD_STRETCH_NW
9	BD_STRETCH_SW
10	BD_MOVE_ALL
11	BD_SLIDE_H
12	BD_SLIDE_V

Description: *Continued*

Attachment types allow either one or two dimensions to be stretched in conjunction with the mouse, or for the entire window to be moved along with the mouse. Also, the slider attachments move the slider within a scroll bar.

The vendor can use a rubber band type of border to indicate the attached movement, move an entire window, or use any other visual indication of the actual movement of the attached window.

In a typical scenario in which MS_ATTACH is used, the mouse is moved over some widget, such as the caption bar, the operator presses down on a mouse button, moves the mouse and releases the button.

When the button is pressed, a mouse event is sent to the host computer. As it may take a while for the event to be recognized by the host computer, the event will contain the mouse's position at the time of the click. This is now the anchor point for the attach.

In the meantime, the operator is probably moving the mouse. When the host computer sends the MS_ATTACH command, the mouse is probably somewhere other than the original anchor point.

The terminal then compares the anchor point with the mouse's current position and adjust the attached window as if the mouse was attached at the anchor point and then moved.

Reply: [None]

Direction: Host ⟶ Terminal

MS_BOUND P_1=205

Group: 2

Sequence: AW_WCI 205 ; <Status>; <Bound_Type>; <X>; <Y>; <Width>; <Height> w

Description: Sets hard or soft mouse boundaries. Coordinates are relative to the physical terminal screen. Note that a maximum of one hard and one soft boundary may exist at any time. Subsequent MS_BOUNDs replace current ones of the same type. If <X>, <Y>, <Width> or <Height> are omitted, their values remain unchanged. Boundary is the space between character cells formed to the outside of the rectangle specified by the parameters.

P_2	<Status>
1	Set
2	Clear

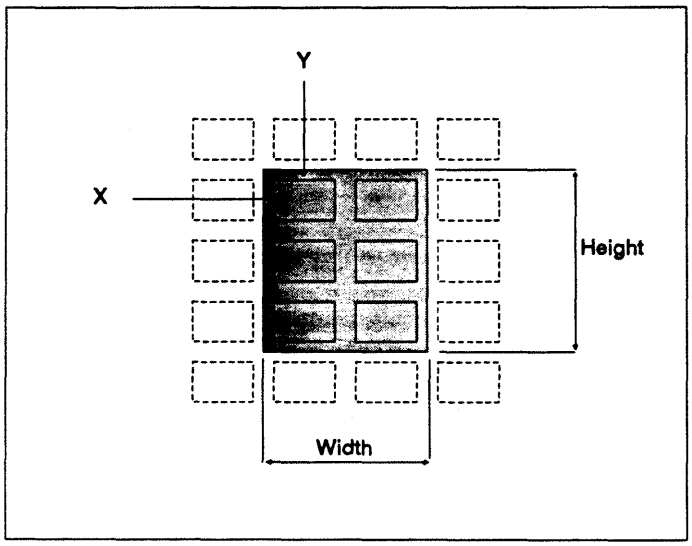
P_3	<Bound_Type>
1	MS_BSOFT
2	MS_BHARD

MS_BSOFT indicates a boundary which triggers a MS_EVENT when crossed in any direction.

MS_BHARD indicates a boundary which the mouse cannot cross, except to enter the boundary area, in which case the mouse cursor stops and no MS_EVENT is output.

Reply: [None]

Direction: Host ⟶ Terminal



MS_BOUND - between Rows and Columns

MS_ENQ P_1=209

Group: 2

Sequence: AW_WCI 209 w

Description: Perform a mouse enquiry.

Reply: [MS_EVENT] (Event Type = MS_STATUS)

Direction: Host ⟶ Terminal

MS_EVENT P_1=213

Group: 2

Sequence: AW_WCI 213 ; <Event_Type>; <P_x>; <P_y>; <Elapsed_Time>; <W_Handle>; <Widget>; <Button_Status>;... ; <Button_Status>; <Key_Modifier>w

Description: Report a mouse event. Mouse cursor coordinates are reported for the location of the mouse hot spot at the time of the report. Elapsed time is an integer number representing tenths of a second since the last mouse event report (e.g 25 is equivalent to 2.5 seconds, sticks at 10 seconds). There are two timers: One timer is assigned only to the button events and the other is assigned to all other events. This eases the timing of button events for the host computer. The first event contains the maximum times value.

The Button Status indication is a series of single digit numbers which represents the status of the mouse buttons. The number of parameters depends on the number reported in the MS_RCONFIG report. This allows other pointing devices which have more than the standard 2 or 3 found on mice. In addition, if devices evolve to have multi-state buttons, the parameters will be allowed values other than 0 and 1.

The widget parameter indicates which widget the mouse cursor hotspot is located over. Widget may be null, and will always be null for terminals which do not support Group 3 operation.

W_Handle is omitted when the mouse is over wallpaper. W_Handle is reported when the mouse is over the client or any widget including the border. MS_CLIENT_ENTER and MS_CLIENT_LEAVE indicate entry and departure of a window's client area.

When the display is changed by AlphaWindow commands, MS_EVENT reports will reflect hte new screen state at the mouse location.

See Event_type, Button Status, and Widget descriptions on the following pages.

P_2	<Event_Type>
1	MS_BUTTON_UP
2	MS_BUTTON_DOWN
3	MS_MOTION
4	MS_WIDGET
5	MS_BOUNDCROSS
6	MS_CLIENT_ENTER
7	MS_CLIENT_LEAVE
8	MS_STATUS

P_n	<Button Status>
1	MS_BUTTON_UP
2	MS_BUTTON_DOWN

P_9, P_{10}, $P_{etc.}$ *	<Key_Modifier>
1	None
2	Ctrl
4	Shift
8	Alt (Left)
16	Alt (Right)
* *Depends on numbers of Buttons.*	

Note: Values for Key_Modifier can be added together in any combination. For example: 6=Ctrl + Shift.

P_7	<Widget>
1	WT_NULLWIDGET
2	WT_MAX ,Maximize Button
3	WT_MIN ,Minimize Button
4	WT_RESTORE ,Restore Button
5	WT_MENU ,Control Menu Button
6	WT_SIZEN ,North window size widget
7	WT_SIZES ,South window size widget
8	WT_SIZEE ,East window size widget
9	WT_SIZEW ,West window size widget
10	WT_SIZENW ,North-west window size widget
11	WT_SIZENE ,North-east window size widget
12	WT_SIZESW ,South-west window size widget
13	WT_SIZESE ,South-east window size widget
14	WT_HSCROLL1 ,Left scroll arrow
15	WT_HSCROLL2 ,Left of slider
16	WT_HSCROLL3 ,Slider
17	WT_HSCROLL4 ,Right of slider
18	WT_HSCROLL5 ,Right scroll arrow
19	WT_VSCROLL1 ,Top scroll arrow
20	WT_VSCROLL2 ,Above slider
21	WT_VSCROLL3 ,Slider
22	WT_VSCROLL4 ,Below slider
23	WT_VSCROLL5 ,Bottom scroll arrow
24	WT_CAPTION ,Title Bar
25	WT_BORDER

Reply: [None]

Direction: Host ⟵ Terminal

Group: 2

Sequence: AW_WCI 217 w

Description: Requests the mouse hardware configuration.

Reply: [MS_RCONFIG]

Direction: Host ⟶ Terminal

MS_MODE P_1=221

Group: 2

Sequence: AW_WCI 221 ; <Mode>;...; <Mode> w

Description: Set the mouse event reporting mode. Widget crossing events are only valid for Group 3 terminals. Modes are additive.

P_n	<Mode>
1	MS_DISABLE
2	MS_CLICKS
3	MS_MOTION
4	MS_WIDGET
5	MS_BOUNDCROSS
6	MS_CLIENT_ENTER/LEAVE
7	MS_SHORT_EVENTS

Reply: MS_EVENT (Type = MS_STSTUS)

Direction: Host ⟶ Terminal

Note: MS_DISABLE completely disables the mouse.

MS_MOVE P_1=225

Group: 2

Sequence: AW_WCI 225 ; <X>; <Y> w

Description: Move mouse cursor. X and Y are the physical terminal coordinates for the cursor hotspot.

Reply: [None]

Direction: Host ⟶ Terminal

MS_RCONFIG P_1=229

Group: 2

Sequence: AW_WCI 229 ; <Buttons> w

Description: Reply to MS_GCONFIG. If "Buttons" is zero then no mouse is attached to the terminal. Otherwise it is the number of buttons that the mouse provides.

Reply: [None]

Direction: Host ⟵ Terminal

MS_SHORT_EVENT P_1= 2

Group: 2

Sequence: AW_WCI 2; <Px>;<Py>;<W_Handle>;<Type>w

Description: High performance mouse event utilizing the minimum number of characters. If any of Px, Py, or W_Handle remains the same from the previous MS_MOTION / MS_SHORT_EVENT then they can be omitted.

P_6	<Type>
	MS_MOTION
	MS_CLIENT_ENTER
	MS_CLIENT_LEAVE

erminal

MS_STYLE P_1=233

Group: 2

Sequence: AW_WCI 233 ; <Cursor_Style> w

Description: Set mouse cursor style.

P_2	<Cursor_Style>	Description
1	MS_ARROW	**Free cursor - pointer**
2	MS_INVISIBLE	No cursor
3	MS_IBEAM	Data entry cursor
4	MS_WAIT	Hourglass - wait for process to complete
5	MS_CROSS	"Move" window cursor
6	MS_UPARROW	Upward pointing arrow
7	MS_SIZE	Small box on corner of larger box for sizing
8	MS_SIZENWSE	Diagonal resize (south to east)
9	MS_SIZENESW	Diagonal resize (south to west)
10	MS_SIZEWE	Resize (left to right)
11	MS_SIZENS	Resize (up to down)

Reply: [None]

Direction: Host ⟶ Terminal

AW_SDECORATION P_1=261

Group: 3

Sequence: AW_WCI 261 ; <W_Handle>; <Decoration>;....; <Decoration> w

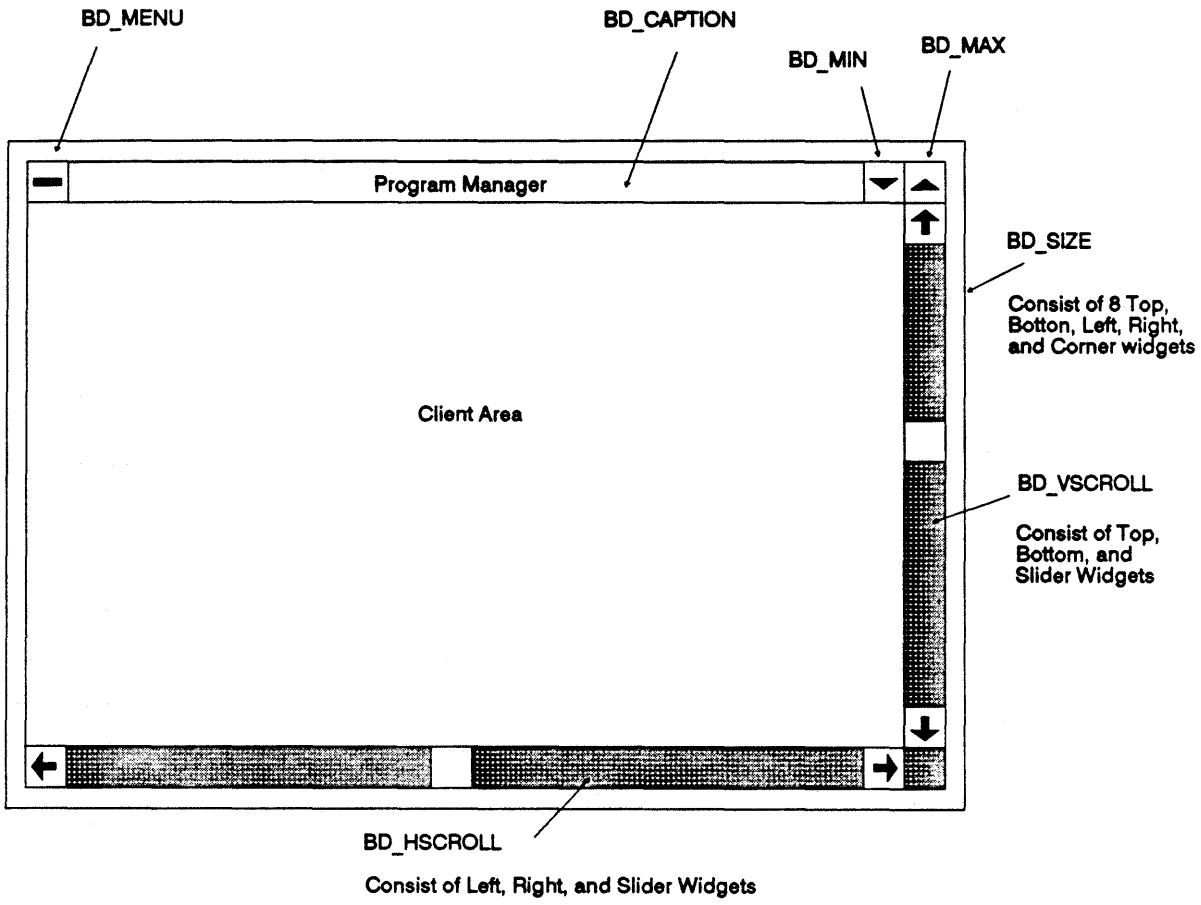
Description: Specifies border decoration requirements. The decorations are additive. Note that an AW_SDECORATION request always overrides a conflicting AW_SBORDER request if the terminal supports Group 3 functionality (Example: BS_NONE will be overridden by BD_SIZE). A decoration can consist of one or more widgets

P_n	<Decoration>
1	BD_NONE
2	BD_MAX
3	BD_MIN
4	BD_SIZE_NORM
5	BD_SIZE_BOLD
6	BD_RESTORE
7	BD_MENU
8	BD_VSCROLL
9	BD_HSCROLL
10	BD_CAPTION

Reply: [None]

Direction: Host ⟶ Terminal

Note: BD_SIZE means a window border with 8 widgets which can be used for re-sizing. This would override any border specified by AW_SBORDER.



Sample AlphaWindow Layout (not to scale)

AW_ADDCREDIT $P_1 = 3$

Group: 4

Sequence: AW_WCI 301 ; <VT Handle>; <Credits> w

Description: Sent between the AlphaWindow terminal and the host to allow characters to be sent. One credit corresponds to a block of 32 bytes.

VT Handle is the Circuit ID for the logical circuit.

Credits is the number of 32 byte blocks of characters allowed to be sent.

Reply: [None]

Direction: Host ⟷ Terminal

Note: AlphaWindow commands do not use credits.

Note: Default for credits is 4.

AW_CREDITENQ P_1=309

Group: 4

Sequence: AW_WCI 309 ; <VT_Handle> w

Description: Requests the number of credits available to the session, (i.e. how many more full blocks can the transmitting device send to the specified session.)

Reply: [AW_RCREDIT]

Direction: Host ⟷ Terminal

AW_RCREDIT P_1=313

Group: 4

Sequence: AW_WCI 313 ; <VT Handle>; <Credits> w

Description: Response to the AW_CREDITENQ command. Indicates how many Credits are available for the specified Circuit ID. Response is rounded downward.

Reply: [None]

Direction: Host ⟷ Terminal

Note: 32 byte blocks

See Also: AW_CREDITENQ

AW_ZEROCREDIT P_1=317

Group: 4

Sequence: AW_WCI 317 ; <VT Handle> w

Description: Set the number of credits remaining for this session to zero.

Reply: [None]

Direction: Host ⟷ Terminal

AlphaWindow
Application Protocol

Revision 1.0

30 May, 1992

Display Industry Association
1007 Elwell Court Suite B
Palo Alto
CA 94303
USA

Tel: 415-967-6888
Fax: 415 960 3522

A Display Industry Association standard implies a concensus of those substantially concerned with its scope and provisions. This standard is intended as a guide to aid the manufacturer, the consumer and the general public. The existence of a Display Industry Association standard does not in any respect preclude anyone, whether he has approved the standard or not, from manufacturing, marketing, purchasing, or using products, processes, or procedures not conforming to the standard. Display Industry Association standards are subject to periodic review and users are cautioned to obtain the latest editions.

CAUTION NOTICE: This Display Industry Association standard may be revised or withdrawn at any time. Purchasers of Display Industry Association standards may receive current information on all standards by calling or writing the Display Industry Association.

Published by:

Display Industry Association
1007 Elwell Court Suite B
Palo Alto
CA 94303
USA

Introduction

The AlphaWindow Application Protocol defines the way in which applications should use the AlphaWindow Terminal Protocol to communicate with the terminal or window manager. It also determines the ways in which a window manager is allowed to modify application commands to maintain its window management policy.

All of the commands in this document have the same format as defined in the AlphaWindow Terminal Specification unless otherwise indicated. Any differences in the meaning or interpretation of commands and parameters are described. If no differences are noted, the command operates in the same way as described in the terminal specification.

Overview

As described in the AlphaWindow Software Architecture, most applications will connect to an AlphaWindow terminal via a window manager. The window manager allows the user to run multiple applications at the same time.

Definitions

Client A single process which connects to the window manager or directly to the terminal to display one or more windows. Multiple clients may be simultaneously displaying multiple windows on a single AlphaWindow terminal by using a window manager to multiplex several client-terminal conversations over the single host-terminal connection.

Virtual Terminal

A single instance of a terminal emulation resident within an AlphaWindow terminal. The virtual terminal is visualised on the display by creating one or more windows which display all or part of the virtual screen. Often abbreviated to VT.

Virtual terminals and windows are identified within the protocol by unique numbers called handles. An individual client may only refer to virtual terminals and windows created by itself.

Window

A rectangular area which maps part of a virtual terminal onto the physical screen. A window may have a border drawn around it, and some AlphaWindow terminals support the ability to draw a set of decorations around a window as well. In summary, a single physical screen may have, at any time, one or more virtual terminals associated with it, each of which may be displayed via one or more windows. All the user ever sees are the portions of the windows which are visible at any time. That is, those parts which are not partially or wholly occluded by other windows or clipped by the physical screen boundary.

Window manager

The process which mediates access to the AlphaWindow terminal between several clients and which ensures that keyboard and mouse input are delivered to the correct client. A window manager will typically provide ways for the user to move, resize and restack windows, and launch new applications.

Special Characters

The special characters AW_WCI, AW_MPI, AW_LITERAL, AW_XON, AW_XOFF and AW_BREAK function in the way defined in the AlphaWindow Protocol.

Windowing

The AlphaWindow Protocol supports a single level hierarchy of windows. This means that child windows have to be visualised by a client drawing into the window. Obviously the visual appearance of this will depend on the terminal emulation being used.

Keyboard Focus

The keyboard focus policy to be adopted is at the discretion of the window manager. The policy will not require client involvement unless any protocol extensions are in use.

Commands

Each command is dealt with in sequence, followed by a list of the responses which may be received by clients. The following notes apply generally to all commands.

1. No application may directly change any property of a virtual terminal or window not created by itself. Some commands do have side effects which affect other applications. An example of this is the AW_STACK command which alters the global stacking order of windows.

   The window manager should police this restriction by checking the parameters of commands received from applications. In cases where an illegal command is received which demands a response (for example AW_OPEN_WIN on a VT which does not belong to the client) the response should be synthesised by the window manager in such a way as to indicate failure (for example by returning a zero window handle in the AW_RWIN). When an illegal command does not have a corresponding response it should be ignored.

2. The window manager may always intercept or modify an application's request in order to avoid its policy being breached. An example of this would be a window manager which implemented a tiled display modifying or even refusing an AW_SGEOM command to maintain the tiling.

How Commands are Described

Each command is described in the format shown below.

AW_COMMAND P1 = n

Notes: Differences in the effect or meaning of the command when a window manager is present.

Client Commands

A new command group and several new commands have been defined to allow explicit communication between clients and the window manager and other future protocol extensions. The definition of this command is extensible and allows the details to be worked out as necessary by the API Working Group. The purpose of the command is to allow the exchange of information which is outside of the scope of the existing protocol.

The following commands may be used by all clients.

Group 1

AW_BEGIN P1 = 7

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

AW_CLOSE_WIN P1 = 9

Reply: None

Notes: W_Handle must identify a window created by this client. The command has no effect otherwise.

AW_CREATE_VT P1 = 13

Reply: AW_RVT

Notes: An application wishing to size a VT or window relative to the screen may use AW_GDISPSZ to discover the current display size.

AW_DA P1 = 17

Reply: AW_RDA

Notes: A window manager which implements the Extensions group (group 5) is required to add group 5 to the AW_RDA response if the terminal itself does not implement the group. The window manager is then responsible for ensuring that no group 5 commands are passed through to a terminal which does not implement that group.

AW_DELETE_VT P1 = 25

Reply: None

Notes: VT_Handle must refer to a VT created by this client. The command has no effect otherwise.

AW_ENABLE_GROUP P1 = 33

Reply: None

Notes: Although the underlying AlphaWindow command is global in its effect, the window manager will make it look as though the command only affects this client. Thus, a client which has not enabled group 2 will never receive any group 2 commands even if another client has enabled group 2. The window manager will filter the data stream to enforce this. A client will not normally request the use of group 4 when a window manager is present since flow control between the terminal and host will be handled by the window manager.

AW_EXIT P1 = 37

Reply: AW_REXIT

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

AW_GDISPSZ P1 = 41

Reply: AW_RDISPSZ

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

AW_GEMUL P1 = 43

Reply: AW_REMUL

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

AW_GGEOM P1 = 45

Reply: AW_RGEOM

Notes: W_Handle must refer to a window created by this client or the AW_RGEOM reply will contain a zero window handle to indicate an error.

AW_OPEN_WIN P1 = 53

Reply: AW_RWIN

Notes: VT_Handle must refer to a VT created by this client or the AW_RWIN reply will contain a zero window handle to indicate an error.

AW_SBORDER P1 = 81

Reply: None

Notes: W_Handle must refer to a window created by this client or the command will have no effect.

AW_SGEOM P1 = 97

Reply: None

Notes: The window manager is permitted to ignore or modify any or all of the parameters to this command in order to maintain whatever display management policy it chooses to impose.

AW_STACK P1 = 105

Reply: None

Notes: W_Handle must refer to a window created by this client. If it does not the command will be ignored.

AW_TITLE P1 = 109

Reply: None

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

AW_VISIBILITY P1 = 117

Reply: None

Notes: W_Handle must refer to a window created by this client or the command will be ignored.

Group 2

MS_ENQ P1 = 209

Reply: MS_EVENT (with event type MS_STATUS)

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

MS_GCONFIG P1 = 217

Reply: MS_RCONFIG

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

MS_MODE P1 = 221

Reply: None

Notes: Not all modes may be used by clients. The mode MS_WIDGET will be ignored as it is reserved for the window manager. Although the underlying AlphaWindow command is global in its effect, the window manager should make it look as though this command only affects this client.

MS_MOVE P1 = 225

Reply: None

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

MS_STYLE P1 = 233

Reply: None

Notes: Although the underlying AlphaWindow command is global in its effect, the window manager should ideally make it look as though this command only affects this client.

Group 3

AW_SDECORATION P1 = 261

Reply: None

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

Group 4

There are no group 4 commands which are normally used by clients when a window manager is present.

Group 5

This new command group contains commands which allow an application to determine the terminal and window manager configuration, and a command to allow extensions to the protocol. None of these commands should be used until the application has determined that this group is supported by the window manager/terminal to which it is connected. The group will then need to be enabled using AW_ENABLE_GROUP.

Protocol extensions may be provided by both the terminal and the window manager. The list of extensions supported in the current environment is returned to the application in the AW_RID response detailed below. An application may not attempt to use an extension without first ascertaining that the extension is available. Applications which make use of very specialised extensions may not be able to run in an environment where those extensions are not available, but most applications should not rely exclusively on an extension to be able to run.

Command codes 400 to 409 are allocated to group 5.

AW_ID P1 = 401

Sequence: AW_WCI 401 w

Description: Device identification request. The AW_RID reply carries information about the presence of a window manager, extensions and the terminal manufacturer.

Reply: AW_RID

Direction: Application -> Window Manager/Terminal

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

AW_EXTENSION P1 = 405

Sequence: AW_WCI 405; <Code>; <Tag>; <P_4>; ...; <P_n> w <Data> ST

Description: Used to invoke an extension command. <Code> is a number which identifies the extension command. Ranges of commands for use by different extensions will be agreed and published by the DIA. <Tag> is a number to be used as the <Tag> in any reply to this command. This allows an application with several outstanding AW_EXTENSION replies to identify which command a particular reply relates to. If <Tag> is zero or omitted then the corresponding tag in any response may be omitted. Parameters P_4 to P_n are all optional and the number and meaning of them is defined by the extension. <Data> allows a single string parameter to be passed. Again, the format and interpretation of this is extension dependent.

Reply: Extension dependent, replies are carried in another AW_EXTENSION or AW_SHORT_EXTENSION command with a defined command code. Not all extension commands will cause a reply.

Direction: Bidirectional

Notes: This command is bidirectional, so that the window manager, or an extension resident in the terminal may send information to the application asynchronously. An example of this might be an extension command from the window manager to inform the application that the user has selected the Close option from the window manager's menu for a particular window. This would be dispatched asynchronously to the application when the event occurs.

AW_SHORT_EXTENSION P1 = 4

Sequence: AW_WCI 4; <Code>; <Tag>; <P_4>; ...; <P_n> w

Description: Used to invoke an extension command which does not require a string parameter. All other aspects of this command are the same as for AW_EXTENSION.

Reply: Extension dependent, replies are carried in another AW_EXTENSION or AW_SHORT_EXTENSION command with a defined command code. Not all extension commands will cause a reply.

Direction: Bidirectional

Client Responses

These are the responses which may be generated by the client commands listed above. Certain of these responses may also be received unsolicited.

Group 1

AW_RBEGIN P1 = 55

Reply to: AW_BEGIN

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

AW_RDA P1 = 59

Reply to: AW_DA

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

AW_RDISPSZ P1 = 61

Reply to: AW_GDISPSZ

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

AW_REMUL P1 = 64

Reply to: AW_GEMUL

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

AW_REXIT P1 = 63

Reply to: AW_EXIT

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

AW_RGEOM P1 = 65

Reply to: Unsolicited

Notes: W_Handle will only ever refer to a window created by this client, or it may be zero to indicate an incorrect AW_GGEOM command. This reply may be received if the window has been resized by the user or an AW_SGEOM request has been modified by the window manager, or in reply to an AW_GGEOM command.

AW_RVT P1 = 73

Reply to: AW_CREATE_VT

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

AW_RWIN P1 = 77

Reply to: AW_OPEN_WIN

Notes: The window handle in this response will be zero if the terminal was unable to create a new window or if an incorrect VT handle was given in the AW_OPEN_WIN command.

Group 2

MS_EVENT P1 = 213

Reply to: MS_ENQ and unsolicited when the user employs the mouse.

Notes: A client will only receive MS_EVENT commands for mouse events relating to windows which it has created.

MS_RCONFIG P1 = 229

Reply to: MS_GCONFIG

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

MS_SHORT_EVENT P1 = 2

Reply to: Unsolicited when the user employs the mouse.

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

Group 3

There are no group three responses.

Group 4

There are no group four responses for normal use by clients.

Group 5

AW_RID P1 = 409

Sequence: AW_WCI 409; <Connection_Type>; <Code>; <Version>; ... ; <Code>; <Version> w <Data> ST

Description: Reply to a device identification request. <Connection_Type> specifies whether the application is directly connected to the terminal or whether a window manager is running. It may take the following values:

P_2	<Connection_Type>
1	CT_TERMINAL - direct terminal connection
2	CT_WMGR - window manager running

The remaining numeric parameters form pairs which specify which extensions are present and their version numbers. <Code> is the command code as registered with the DIA and <Version> is the version number multiplied by 100.

The <Data> string is formatted into a number of fields, each separated by a forward slash character. Any field may be left empty if the information is not known. The fields are described in the table below:

Field	Meaning
1	Terminal manufacturer
2	Terminal product name
3	Terminal firmware release
4	Window manager author
5	Window manager product name
6	Window manager software release

An example <Data> string would be:

```
Acme Corp/Acme 220/1.3/WM Inc/AW-WM/2.2
```

Reply: None

Direction: Window Manager/Terminal -> Application

Notes:

Window Manager Only Commands and Responses

The following commands are normally used only by window managers or turnkey applications which take direct control of the terminal. They concern global terminal state and commands and responses required by the window manager for layout information. In addition, certain mouse functionality such as attaching a window to the mouse is reserved for window managers.

Group 1

AW_ATTENTION P1 = 1

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

AW_DATA P1 = 21

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

AW_DESELECT P1 = 29

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

AW_FREEZE_REF P1 = 38

Notes: A window manager will ignore this command.

AW_GBORDER P1 = 39

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

AW_RATTN P1 = 54

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

AW_RBORDER P1 = 57

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

AW_SDISPSZ P1 = 85

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

AW_SELECT P1 = 89

Notes: What happens if all or part of the selected text is scrolled out of the VT (the spec doesn't say). This should be tightened up.

AW_SEND P1 = 91

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

AW_SETATTN P1 = 93

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

AW_SKBD P1 = 101

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

AW_TITL_HILIT P1 = 111

Notes: A window manager will ignore this command.

AW_TRACK P1 = 113

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

Group 2

MS_ATTACH P1 = 201

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

MS_BOUND P1 = 205

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

Group 3

There are no group 3 commands reserved for use by the window manager.

Group 4

AW_ADDCREDIT P1 = 3

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

AW_CREDITENQ P1 = 309

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

AW_RCREDIT P1 = 313

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

AW_ZEROCREDIT P1 = 317

Notes: This command functions as specified in the AlphaWindow Terminal Specification.

AlphaWindow Library
Application Programming Interface
Specification

Revision 1.0

30 May, 1992

Display Industry Association
1007 Elwell Court Suite B
Palo Alto
CA 94303
USA

Tel: 415-967-6888
Fax: 415 960 3522

A Display Industry Association standard implies a concensus of those substantially concerned with its scope and provisions. This standard is intended as a guide to aid the manufacturer, the consumer and the general public. The existence of a Display Industry Association standard does not in any respect preclude anyone, whether he has approved the standard or not, from manufacturing, marketing, purchasing, or using products, processes, or procedures not conforming to the standard. Display Industry Association standards are subject to periodic review and users are cautioned to obtain the latest editions.

CAUTION NOTICE: This Display Industry Association standard may be revised or withdrawn at any time. Purchasers of Display Industry Association standards may receive current information on all standards by calling or writing the Display Industry Association.

Published by:

Display Industry Association
1007 Elwell Court Suite B
Palo Alto
CA 94303
USA

Introduction

This document describes a C language API known as AWlib for use by application and toolkit developers. It gives access to the full functionality of the AlphaWindow Application Protocol for the creation and control of windows and the use of the mouse. It does not control or restrict what is displayed by an application within windows which it creates. Applications are, however, only allowed to work with windows which they themselves have created.

The underlying AlphaWindow Protocol, and hence the AlphaWindow Application Protocol both contain optional elements such as mouse support and window decoration. AWlib contains a number of functions which enable applications to determine which, if any, of those options are present in a particular environment. To guarantee application portability and inter-operability it is important that developers take advantage of these AWlib facilities. This will also ensure that applications use AlphaWindow terminals to their best possible effect.

AWlib gives full access to all of the functionality of an AlphaWindow terminal, including those operations designed for use by window managers. Those functions in this specification which are marked with a dagger (†) would not normally be used by applications when running in conjunction with a window manager. They are specified in order that an application may use them to present a fully functional windowing interface when running stand-alone in the absence of a window manager.

Overview

AWlib provides a simple, low-level interface to the features of an AlphaWindow terminal. It is intended for use both by applications which will normally run with a window manager and by turnkey software which assumes sole ownership of the terminal.

An AlphaWindow terminal provides multiple concurrent terminal emulations known as *virtual terminals* (VTs). These virtual terminals are visualised on the terminal screen via a single level of windowing. Each window has an associated virtual terminal and displays all or part of the screen of that terminal. The window may be smaller than the virtual terminal and can then be panned around the VT either under user or application control. A window may be *minimized*, when the image of the window is replaced on the screen by a small icon. A window may be displayed with several styles of border and many terminals support the ability to add extra decorations to the edge of the window such as resize handles, a title bar and scroll bars. The part of the window inside the border or decorations is known as the client area.

The terminal may posess a mouse. If so, the application can ask for mouse events to be delivered when the mouse is moved, clicked or crosses a window boundary. Other input, including keyboard input is delivered to the application via other types of event.

Types

A number of new types are defined as part of the AWlib API. The new types which are used throughout the library are listed in Appendix A. A number of structures which are only used with one or two functions are defined in the text.

Coordinates

All coordinates and dimensions are measured in characters. All coordinates related to windowing begin from one. The coordinate system used by an individual emulation within a virtual terminal is, of course, specified by the terminal manufacturer. All coordinate parameters within AWlib are of type **Position**, and all other measurements such as window size are of type **Dimension**. As is customary in many windowing API's, the origin of a window refers to the top left hand corner of the window's client area.

Error Handling

Errors are reported using error return codes from functions. Many functions return AW_OK on successful completion and AW_ERROR if a problem was encountered. All other functions which return other types have a distinguished return value which indicates an error. More information about the error may be obtained by looking at the value of an error code variable *aw_errno* declared in <awlib.h> as follows.

```
extern int aw_errno;
```

Appendix B lists the error types which have been defined as part of this API.

How Functions are Described

Each function in AWlib is described in the format shown below:

AWLibFunction()

```
AW_Status AWLibFunction(VT_Handle   vt,
                        int         arg2,
                        ...)
```

Description: A summary of the effect of the function and the purposes of the arguments.

Returns: Possible return values and their meanings.

Errors: Possible values of *aw_errno* after an error return. Individual AWlib implementations may also return other values.

Commands: AlphaWindow Application Protocol commands used by this function (subject to any implementation specific optimisations).

Initialisation

AWInit()

```
AW_Status AWInit(char        aw_wci,
                 char        aw_mpi,
                 char        aw_literal,
                 char        aw_xon,
                 char        aw_xoff,
                 char        aw_break)
```

Description: This function must be called before any other AWlib function. It initialises various internal data structures. The parameters may be used to set the values to be used by the library for the special characters within the AlphaWindow Protocol. To use the default value as defined in the AlphaWindow Terminal Specification pass zero as the parameter value. The effect of calling AWInit() more than once is undefined.

Returns: AW_OK if initialisation was completed or AW_ERROR if a problem was detected.

Errors: AW_NO_MEMORY
AW_TIMED_OUT

Commands: AW_BEGIN, AW_RBEGIN, AW_DA, AW_RDA

AWFinish()

```
AW_Status AWFinish(void)
```

Description: This function is called to indicate to the terminal that the use of AlphaWindow functionality by this application has finished. All virtual terminals and windows will be destroyed and any pending events deleted. After this function, AWResume() must be called before any other AWlib functions may be used.

Returns: AW_OK if successful or AW_ERROR if a problem was detected..

Errors: AW_TIMED_OUT

Commands: AW_EXIT, AW_REXIT

AWResume()

```
AW_Status AWResume(void)
```

Description: This function is called to indicate to the terminal that the application wishes to resume the use of AlphaWindow functionality. The effect of calling AWResume() at any time other than immediately after AWFinish() is undefined.

Returns: AW_OK if successful or AW_ERROR if a problem was detected..

Errors: AW_TIMED_OUT

Commands: AW_BEGIN, AW_RBEGIN

Terminal Capabilities

The functions in this section allow an application to determine various facts about the terminal such as the display size or whether there is a mouse attached.

AWIsMouseSupported()

```
AW_Boolean AWIsMouseSupported()
```

Description: This function queries the terminal to find out whether it supports the use of a mouse.

Returns: AW_OK if the terminal has a mouse or AW_ERROR if there is no mouse support or a problem was detected. If a problem was detected then aw_errno will be set to an appropriate value other than AW_NO_ERROR.

Errors: AW_NO_MEMORY

Commands: AW_DA, AW_RDA

Note: There is no AWEnableMouse() function in AWlib since that is part of the functionality of AWSetMouseMode().

AWIsDecorationSupported()

```
AW_Boolean AWIsDecorationSupported()
```

Description: This function queries the terminal to find out whether window decorations are supported.

Returns: AW_OK if window decorations are supported or AW_ERROR if there are no window decorations or a problem was detected. If a problem occurred then aw_errno will be set to an appropriate value other than AW_NO_ERROR.

Errors: AW_NO_MEMORY

Commands: AW_DA, AW_RDA

AWEnableDecoration()
AWDisableDecoration()

```
AW_Status AWEnableDecoration(void)
AW_Status AWDisableDecoration(void)
```

Description: These functions are called to respectively enable or disable window decoration for this client.

Returns: AW_OK if decorations were successfully enabled, or AW_ERROR if there was a problem.

Errors: AW_NO_DECORATIONS

Commands: AW_ENABLE_GROUP

AWListEmulations()

```
char **AWListEmulations(void)
```

Description: Returns an array of the names of the terminal emulations supported by the user's terminal. The array is a list of strings (character pointers) terminated by a null pointer and will always contain at least one element. The array is a data structure belonging to the library and should not be modified by applications. An example of the use of this function is:

```
char **list;
int i;

list = AWListEmulations();

i = 0;
while (list[i] != NULL) {
      printf("Emulation %d is %s\n", i, list[i]);
      i++;
}
```

The first name in this list is the default emulation used for a VT when no other emulation is specified.

Returns: A pointer to the array of emulation names, or a null pointer to indicate an error.

Errors: AW_TIMED_OUT
AW_NO_MEMORY

Commands: AW_GEMUL, AW_REMUL

AWGetTerminalInfo()

```
typedef struct {
      int               code;       /* Extension code number */
      int               version;    /* Extension version number */
} ExtensionDesc;

typedef struct {
      Terminal_Type     type;       /* Terminal or window manager? */
      char              *term_maker;/* Manufacturer of terminal */
      char              *term_name; /* Product name of terminal */
      char              *term_release;    /* Firmware release */
      char              *wm_maker;  /* Manufacturer of window manager */
      char              *wm_name;   /* Product name of window manager */
      char              *wm_release;      /* Window manager release */
      int               num_extensions;   /* Number of extensions */
      ExtensionDesc     *extensions;      /* List of extensions */
} AWTerminalInfo;

AWTerminalInfo *AWGetTerminalInfo(void)
```

Description: This function queries the terminal/window manager to find out the information in the structure above. The structure member *extensions* is a pointer to a list of extension descriptors that specify which extensions are supported in this environment. The length of the list is given in the *num_extensions* member. The data structure returned by this function belongs to the library and should not be modified by the application.

Returns: A pointer to a filled-in AWTerminalInfo structure if the call was successful or a null pointer if there is no extension support or a problem was detected.

Errors: AW_NO_EXTENSION

Commands: AW_ID, AW_RID

AWGetDisplaySizes()

```
/* All measurements in this structure are in characters */

typedef struct {
      Dimension   icon_width;       /* Width of an icon */
      Dimension   icon_height;      /* Height of an icon */
      Dimension   disp_width;       /* Physical display width */
      Dimension   disp_height;      /* Physical display height */
      Dimension   width_lo;         /* Lowest display width */
      Dimension   width_hi;         /* Highest display width */
      Dimension   height_lo;        /* Lowest display height */
      Dimension   height_hi;        /* Highest display width */
      Dimension   list_count;       /* Size of width and height lists */
      Dimension   *width_list;      /* -> permitted display widths */
      Dimension   *height_list;     /* -> permitted display heights */
} AWDisplaySize;

AWDisplaySize *AWGetDisplaySizes(void)
```

Description: Queries the various size metrics of the terminal display. The data structure returned belongs to the library and should not be modified by applications.

Returns: A pointer to a valid AWDisplaySize structure if the query succeeded and a null pointer otherwise.

Errors: AW_TIMED_OUT

Commands: AW_GDISPSZ, AW_RDISPSZ

†AWSetDisplaySize()

```
AW_Status AWSetDisplaySize(int      width,
                           int      height)
```

Description: This function selects *width* and *height* as the current physical display size for the terminal. Note that a window manager may override a display size setting with its own (presumably the user's) preference.

Returns: AW_OK if the command was successful and AW_ERROR otherwise.

Errors: AW_TIMED_OUT

Commands: AW_GDISPSZ, AW_RDISPSZ AW_SDISPSZ

Virtual Terminal Creation and Deletion

AWCreateVT()

```
VT_Handle AWCreateVT( Dimension          init_width,
                      Dimension          init_height,
                      Dimension          max_width,
                      Dimension          max_height,
                      Private_Hint       private_hint,
                      char               *emulation_type)
```

Description: Creates a new virtual terminal with the initial size specified by *init_height* and *init_width*. If the virtual terminal may be resized to a larger size then the maximum expected size should be passed in *max_height* and *max_width*. *Private_hint* should be set to PH_PRIVATE if this virtual terminal is for use by a window manager and PH_NORMAL otherwise. The terminal may use this hint to implement vendor specific features or optimisations. The parameter *emulation_type* may either be one of the names returned by AWListEmulations() or a null pointer, in which case the terminal's default emulation is used.

Returns: The VT handle to use in future calls, or NULL_VT_HANDLE to indicate an error.

Errors: AW_NO_MEMORY
AW_TIMED_OUT
AW_BAD_EMULATION

Commands: AW_CREATE_VT, AW_RVT

AWDeleteVT()

```
AW_Status AWDeleteVT(VT_Handle   vt)
```

Description: Deletes the given VT and all associated windows.

Returns: AW_OK if the VT was successfully deleted, or AW_ERROR if there was a problem.

Errors: AW_BAD_VT

Commands: AW_DELETE_VT

Window Creation and Deletion

AWOpenWindow()
AWOpenMainWindow()

```
W_Handle AWOpenWindow( VT_Handle          vt,
                       Dimension          width,
                       Dimension          height,
                       Window_Type        win_type,
                       Transient_Type     transient_hint)

W_Handle AWOpenMainWindow( VT_Handle      vt,
                           Dimension      width,
                           Dimension      height)
```

Description: AWOpenWindow() creates a new window onto the VT identified by the handle *vt*. *Height* and *width* are measured in characters. *Win_type* may be WT_MAIN to create a regular window or WT_TRANSPARENT to create a transparent window which only displays its outline. *Transient_hint* may be TF_NORMAL or TF_TRANSIENT to indicate that this window is expected to be short lived. The window is not visible until made so by calling AWRevealWindow().

AWOpenMainWindow() is a convenience function which creates a new window with window type WT_MAIN and transient hint TF_NORMAL.

Returns: The window handle to use in future calls, or NULL_WIN_HANDLE to indicate an error.

Errors: AW_BAD_VT
AW_BAD_TYPE
AW_BAD_TRANSIENT
AW_TIMED_OUT
AW_NO_MEMORY

Commands: AW_OPEN_WIN, AW_SGEOM

AWCloseWindow()

```
AW_Status AWCloseWindow(W_Handle window)
```

Description: Closes the given window. The window handle becomes invalid.

Returns: AW_OK if the window was successfully closed, or AW_ERROR if there was a problem.

Errors: AW_BAD_WINDOW

Commands: AW_CLOSE_WIN

Window Size, Position, Stacking and Update

AWConfigureWindow()

```
AW_Status AWConfigureWindow( W_Handle      window,
                             W_State       wstate,
                             Position      x,
                             Position      y,
                             Dimension     width,
                             Dimension     height,
                             Position      virt_x,
                             Position      virt_y)
```

Description: The given window is configured as specified by the other parameters. *Wstate* may be WS_NORM to place the window in the normal, visible state or WS_MIN to minimize the window and replace it with an icon. *X* and *y* specify the position of the window on the terminal screen, *width* and *height* are the new size of the window, and *virt_x* and *virt_y* specify the origin of the window within the associated VT. Any of these parameters may be left unchanged by passing the value 0.

Note that any of these parameters other than *virt_x* and *virt_y* may be overridden by the window manager. In virtually all cases applications will not care about the actual x and y position of the window. If the actual dimensions are important then the application should call AWGetWindowConf() following this function.

Returns: AW_OK if the window was successfully configured, or AW_ERROR if there was a problem.

Errors: AW_BAD_WINDOW

Commands: AW_SGEOM

†AWResizeWindow()

```
AW_Status AWResizeWindow( W_Handle        window,
                          Dimension       width,
                          Dimension       height)
```

Description: The given window is resized as specified by *width* and *height*. This is a convenience function for use when only the size of a window is to be changed. Note that a window manager is permitted to override the width and height given in this call.

Returns: AW_OK if the window was successfully resized, or AW_ERROR if there was a problem.

Errors: AW_BAD_WINDOW

Commands: AW_SGEOM

†AWMoveWindow()

```
AW_Status AWMoveWindow(W_Handle     window,
                       Position     x,
                       Position     y)
```

Description: The given window is moved as specified by *x* and *y*. This is a convenience function for use when only the position of a window is to be changed. Note that a window manager is permitted to override the coordinates given in this call.

Returns: AW_OK if the window was successfully moved, or AW_ERROR if there was a problem.

Errors: AW_BAD_WINDOW

Commands: AW_SGEOM

AWSetWindowOrigin()

```
AW_Status AWSetWindowOrigin(W_Handle      window,
                            Position      virt_x,
                            Position      virt_y)
```

Description: The virtual position of the given window within it's VT is changed as specified by *virt_x* and *virt_y*. This is a convenience function for use when only the virtual position of a window within the virtual terminal is to be changed.

Returns: AW_OK if the window was successfully repositioned, or AW_ERROR if there was a problem.

Errors: AW_BAD_WINDOW

Commands: AW_SGEOM

AWSetWindowState()

```
AW_Status AWSetWindowState(W_Handle       window,
                           Window_State   state)
```

Description: The state of the window is changed as specified by *state*. This is a convenience function for use when only the state of the window is to be changed.

Returns: AW_OK if the window state was successfully changed, or AW_ERROR if there was a problem.

Errors: AW_BAD_WINDOW

Commands: AW_SGEOM

AWGetWindowConf()

```
/* All coordinates in this structure are measured in characters */

typedef struct {
      Window_State      state;      /* Normal or minimized */
      Position          x;          /* X position on screen */
      Position          y;          /* Y position on screen */
      Dimension         width;      /* Width of window */
      Dimension         height;     /* Height of window */
      Position          virt_x;     /* X origin of window in VT */
      Position          virt_y;     /* Y origin of window in VT */
      Dimension         virt_screen_width;      /* VT width */
      Dimension         virt_screen_height;     /* VT height */
      Dimension         disp_width; /* Current display width */
      Dimension         disp_height;/* Current display height */
      Dimension         caption_width;    /* Max title bar text size */
} AWWindowConf;

AW_Status AWGetWindowConf( AWWindowConf *confp)
```

Description: Returns the current configuration of the given window by filling in the structure pointed to by *confp*. The meanings of the structure members are explained above.

Returns: AW_OK if the configuration was retrieved successfully, or AW_ERROR if there was a problem.

Errors: AW_BAD_WINDOW

Commands: AW_GGEOM, AW_RGEOM

AWStackWindow()

```
AW_Status AWStackWindow(W_Handle    win,
                        Stack_Type  stack)
```

Description: Brings the given window to the front or back of the window stack. *Stack* may take the value SR_PROMOTE to bring the window to the front, or SR_DEMOTE to send it to the back.

Returns: AW_OK if the command was successful, or AW_ERROR if there was a problem.

Errors: AW_BAD_WINDOW

Commands: AW_STACK

AWSetWindowTracking()

```
AW_Status AWSetWindowTracking(W_Handle    win,
                              int         track)
```

Description: Sets the cursor tracking status of the given window. When cursor tracking is enabled, the terminal will ensure that the current cursor position is visible by changing the origin of the window within it's virtual terminal. Horizontal and vertical tracking may be set independently of each other. *Track* is the bitwise or of any of the following flag bits:

Flag	Meaning
CS_NONE	No tracking
CS_HTRACK	Horizontal tracking on
CS_VTRACK	Vertical tracking on

Returns: AW_OK if the tracking was set successfully, or AW_ERROR if there was a problem.

Errors: AW_BAD_WINDOW

Commands: AW_TRACK

†AWFreezeDisplay()
†AWThawDisplay()

```
AW_Status AWFreezeDisplay(void)

AW_Status AWThawDisplay(void)
```

Description: AWFreezeDisplay() causes the refreshing of the terminal screen to be paused and AWThawDisplay() continues refresh and causes any pending display changes to be made. This would typically be used to prevent the display of incomplete results when a number of window management operations are performed in succession.

Returns: AW_OK if the operation was successful, or AW_ERROR if there was a problem.

Errors:

Commands: AW_FREEZE_REF

Other Window Attributes

†AWSetWindowBorder()

```
AW_Status AWSetWindowBorder( W_Handle      window,
                             Border_Style  border)
```

Description: The given window is changed to have the border style specified in *border*. The styles available are BS_THICKNORMAL, BS_THIN, BS_NONE, BS_THICKBOLD and BS_GHOSTOUTLINE. This function only gives a hint to the window manager, which may override the border setting. The next function, AWSetWindowDecoration() interacts with this function since window decorations and border settings interact within the terminal.

Returns: AW_OK if the window was successfully rebordered, or AW_ERROR if there was a problem.

Errors: AW_BAD_WINDOW
AW_BAD_BORDER

Commands: AW_SBORDER

AWGetWindowBorder()

```
/* All dimensions in this structure are measured in characters */

typedef struct {
      Dimension   top;        /* Thickness of top border */
      Dimension   right;      /* ... of right border */
      Dimension   bottom;     /* ... of bottom border */
      Dimension   left;       /* ... of left border */
} Window_Border;

AW_Status AWGetWindowBorder( W_Handle           window,
                             Window_Border      *borderp)
```

Description: This function retrieves the border sizes for the given window. The size information is placed in the Window_Border structure pointed to by *borderp*.

Returns: AW_OK if the information was successfully retrieved, or AW_ERROR if there was a problem.

Errors: AW_BAD_WINDOW

Commands: AW_GBORDER, AW_RBORDER

†AWSetWindowDecoration()

```
AW_Status AWSetWindowDecoration(W_Handle  window,
                                int       flag)
```

Description: This function is called to change the decoration style of a window. *Flag* is either zero to indicate no decoration, or the bitwise or of any of the following flag bits:

Flag bit	Meaning
BD_MAX	Include maximise button
BD_MIN	Include minimise button
BD_SIZE_NORM	Include normal weight resize handles
BD_SIZE_BOLD	Include bold weight resize handles
BD_RESTORE	Include restore button
BD_MENU	Include the system menu button
BD_VSCROLL	Include a vertical scroll bar
BD_HSCROLL	Include a horizontal scroll bar
BD_CAPTION	Include a caption bar

This call only gives hints to the window manager, which is free to ignore or override any of these flags. The flags themselves are defined in <awlib.h>.

Note that this function will override any conflicting window border which has been set using AWSetWindowBorder().

Returns: AW_OK if the decoration was successfully set, or AW_ERROR if there was a problem.

Errors: AW_NO_DECORATION
AW_BAD_WINDOW

Commands: AW_SDECORATION

AWSetWindowTitle()

```
AW_Status AWSetWindowTitle( W_Handle      window,
                            VT_Handle     vt,
                            Position      x,
                            Position      y,
                            Dimension     length)
```

Description: The title of the given window is changed to the *length* characters situated at position (*x*, *y*) in the virtual terminal *vt*. The application draws the title into a VT and then transfers it to the title bar of a window using this function. This approach allows the use of visual attributes and alternate character sets in the title. The title bar must first have been enabled using AWSetWindowDecoration().

A *length* of zero will result in no title being displayed.

Returns: AW_OK if the call succeeded, or AW_ERROR if there was a problem.

Errors: AW_BAD_WINDOW
AW_BAD_LENGTH

Commands: AW_TITLE

†AWHighlightTitle()
†AWUnhighlightTitle()

```
AW_Status AWHighlightTitle(W_Handle       win)

AW_Status AWUnhighlightTitle(W_Handle     win)
```

Description: Sets the highlight state of the given window's title.

Returns: AW_OK if the highlight was set successfully, or AW_ERROR if there was a problem.

Errors: AW_BAD_WINDOW

Commands: AW_TITL_HILIT

AWRevealWindow()
AWHideWindow()

```
AW_Status AWRevealWindow(W_Handle   window)
AW_Status AWHideWindow(W_Handle  window)
```

Description: AWRevealWindow() makes the given window visible on the display. AWHideWindow() removes the window from the display.

Returns: AW_OK if the call succeeded, or AW_ERROR if there was a problem.

Errors: AW_BAD_WINDOW

Commands: AW_VISIBILITY

The Mouse

AWGetMouseConfig()

```
AW_Status AWGetMouseConfig(int    *buttons_return)
```

Description: This function is called to get the number of mouse buttons. The variable pointed to by *buttons_return* will be set to the number of buttons. This may be zero if the terminal does not have a mouse attached.

Returns: AW_OK if the function succeeded, or AW_ERROR if there was a problem.

Errors: AW_NO_MOUSE

Commands: MS_GCONFIG, MS_RCONFIG

AWSetMouseMode()

```
AW_Status AWSetMouseMode(int      flag)
```

Description: This function is called to configure what events will be generated about mouse usage. Flag is either MS_DISABLE to disable mouse reporting entirely, or the bitwise or of any of the following flag bits:

Flag bit	Meaning
MS_CLICKS	Send mouse clicks
MS_MOTION	Send mouse motion
MS_WIDGET	Send clicks on window decorations
MS_CLIENT_ENTERLEAVE	Send enter/leave window events

It is recommended that MS_MOTION is only turned on when required during a user interaction. Setting the MS_WIDGET flag will have no effect if a window manager is running since the window manager will consume those events for its own use.

Returns: AW_OK if the mouse was successfully enabled, or AW_ERROR if there was a problem.

Errors: AW_NO_MOUSE

Commands: MS_MODE

†AWSetMouseCursor()

```
AW_Status AWSetMouseMode(int        style)
```

Description: This function is called to set the mouse cursor picture. *Style* is one of the following:

Style	Meaning
MS_ARROW	Default arrow pointer
MS_INVISIBLE	No cursor
MS_IBEAM	'I' shaped cursor for text entry
MS_WAIT	Hourglass
MS_CROSS	Cross shape used for move operations
MS_UPARROW	Up arrow
MS_SIZE	Small box on corner of larger box
MS_SIZENWSE	Diagonal resize (south to east)
MS_SIZENESW	Diagonal resize (south to west)
MS_SIZEWE	Resize (left to right)
MS_SIZENS	Resize (up to down)

Note that the window manager is free to override this function.

Returns: AW_OK if the picture was successfully changed, or AW_ERROR if there was a problem.

Errors: AW_NO_MOUSE

Commands: MS_STYLE

AWMouseEnq()

```
AW_Status AWMouseEnq(void)
```

Description: This function is called to request an update on the current state of the mouse. After this function is called, a mouse event with the *event_type* member set to MS_STATUS will be returned by a future call to AWNextEvent(). Note that it will not necessarily be the next event returned - there may be several other intervening events.

Returns: AW_OK if the request was successfully sent, or AW_ERROR if there was a problem.

Errors: AW_NO_MOUSE

Commands: MS_ENQ, MS_EVENT

†AWSetMouseBounds()

```
AW_Status AWSetMouseBounds(Bound_Type     bound_type,
                           Position       x,
                           Position       y,
                           Dimension      width,
                           Dimension      height)

AW_Status AWRemoveMouseBounds(Bound_Type  bound_type)
```

Description: An AlphaWindow terminal which supports a mouse can maintain (on a terminal wide basis only) two rectangular mouse boundaries. The soft boundary indicates a boundary which triggers a mouse event when crossed in any direction. The hard boundary defines a region which may only be entered by the mouse. Once inside, the mouse pointer may not be moved out again. No mouse event is generated by a hard boundary. *Bound_type* may be either MS_BSOFT or MS_BHARD. For AWSetMouseBounds() the boundary rectangle is defined by *x*, *y*, *width* and *height*. To leave any of these unchanged, pass 0 as the parameter value. AWRemoveMouseBounds() removes the current boundary definition of the given type.

Note that subsequent boundary settings replace current ones of the same type. In addition, a window manager is permitted to override boundary settings.

Returns: AW_OK if the boundary was successfully set, or AW_ERROR if there was a problem.

Errors: AW_NO_MOUSE

Commands: MS_BOUND

†AWAttachMouse()
†AWDetachMouse()

```
AW_Status AWAttachMouse(W_Handle          win,
                        Position          x,
                        Position          y,
                        Attach_Type       attach)

AW_Status AWDetachMouse(void)
```

Description: AWAttachMouse() attaches the given window to the mouse with anchor point (*x*, *y*). *Attach* controls the type of attachment and hence the precise effect of this function. The attachment may cause the resizing or movement of the window or movement of either of the scrollbars which can decorate a window. If the mouse position is different from the anchor point when the terminal receives this command then the difference is used to adjust the geometry or position of the window or scrollbar. As the mouse is moved by the user the appropriate geometry or position is continuously updated by the terminal. Only one mouse attachment is allowed at any one time over the whole terminal. AWDetachMouse() ends the attachment.

An example of the use of this function would be in a turnkey application which wanted to perform window management. When a mouse button press in the title area of a window was detected (via an event) the application could use AWAttachMouse() to allow the user to move the window. The anchor point would normally be the mouse position given in the mouse event. When the button is released, the mouse would be detached by calling AWDetachMouse().

Returns: AW_OK if the attachment was successful, or AW_ERROR if there was a problem.

Errors: AW_BAD_WINDOW

Commands: MS_ATTACH

Event Handling

Input (both from the keyboard and other sources) is provided to the application in the form of *events*. An event is data generated asynchronously by the AlphaWindow terminal or Window Manager normally as a result of user activity such as typing or moving the mouse. There are ten types of event defined, some of which have a variety of sub-types.

Event Structure

Each type of event has an individual structure defined to specify the parameters of the event. In addition the **AWEvent** structure is defined as a union of all of these individual structures.

```
typedef union {
      int                     type;
      AWKeyboardEvent         keyboard;
      AWMouseEvent            mouse;
      AWGeometryEvent         geometry;
      AWRoutingEvent          routing;
      AWExtensionEvent        extension;
      AWAttentionEvent        attention;
      AWSelectedDataEvent     selecteddata;
      AWSpecialCharEvent      specialchar;
      AWAddCreditEvent        addcredit;
      AWRestoreEvent          restore;
      AWExitEvent             exit;
} AWEvent;
```

An event structure's first member is always the type. This means that the type can always be accessed as shown below:

```
      AWEvent event;
      AWStatus status;

      status = AWNextEvent(&event);

      switch (event.type) {
      case GeometryEvent:
            new_x = event.geometry.x;
            ...

      }
```

Keyboard Input Event

This event is generated when keyboard input or other emulation data is received from the terminal. The type of this event is **KeyboardEvent**.

```
#define AW_MAX_STRING   64

typedef struct {
      int               type;
      int               length;
      char              string[AW_MAX_STRING];
} AWKeyboardEvent;
```

Mouse Event

This event is generated when mouse input is received from the terminal. The type of this event is **MouseEvent**.

```
typedef struct {
      int               type;
      int               event_type;
      Position          x;
      Position          y;
      int               time;
      W_Handle          window;
      Widget_Type       widget;
      int               buttons;
      int               modifiers;
} AWMouseEvent;
```

Geometry Event

When a window is moved, resized, or changes state a geometry event is generated. The event type is **GeometryEvent**.

```
typedef struct {
      int               type;
      W_Handle          window;
      Window_State      state;
      Position          x;
      Position          y;
      Dimension         width;
      Dimension         height;
      Position          virt_x;
      Position          virt_y;
      Dimension         virt_width;
      Dimension         virt_height;
      Dimension         p_width;
      Dimension         p_height;
      Dimension         caption_width;
} AWGeometryEvent;
```

Routing Event

A routing event is generated when a notification is received from the terminal that input is now being sent from a different VT. The event type is **RoutingEvent**.

```
typedef struct {
      int               type;
      VT_Handle         vt;
} AWRoutingEvent;
```

Extension Event

An extension event is reported when an extension command is received by the library. The event type is **ExtensionEvent**.

```
typedef struct {
      int               type;
      int               code;
      int               tag;
      int               int_count;
      int               char_count;
} AWExtensionEvent;
```

The extension command will have zero or more numeric parameters and a string parameter associated with it. To retrieve these, use the function AWGetExtensionParms() described in the section on extensions below.

Attention Event

An attention event is reported when a set of keys which have been registered by an application along with an attention identifier are pressed at the same time. The event type is **AttentionEvent**.

```
typedef struct {
      int               type;
      int               id;
} AWAttentionEvent;
```

Selected Data Event

A selected data event is reported at some time after a call to AWGetSelection(). The event structure contains a pointer to a buffer containing the selected data and a count of the bytes in the buffer. The buffer is owned by the library, and the pointer is only valid until the next call to AWGetSelection(). If an application wishes to preserve the data it should copy it into a buffer of its own. The event type is **SelectedDataEvent**.

```
typedef struct {
      int               type;
      int               count;
      char              *data;
} AWAttentionEvent;
```

Special Character Event

This event is generated when one of the three special characters AW_BREAK, AW_XON and AW_XOFF is received by the library. The event type is **SpecialCharEvent.**

```
typedef struct {
      int               type;
      VT_Handle         vt;
      char              special;
} AWSpecialCharEvent;
```

†Add Credit Event

When an AW_ADDCREDIT command is received, the library will report an event of type **AddCreditEvent.**

```
typedef struct {
      int               type;
      VT_Handle         vt;
      int               credits;
} AWAddCreditEvent;
```

Restore Event

When the library receives an AW_RESTORE command (indicating that the terminal's environment has been corrupted by, for example, a power failure) an event of type **RestoreEvent** is reported. When the event is received, the application should consider all existing window and VT handles invalid.

```
typedef struct {
      int               type;
} AWRestoreEvent;
```

Exit Event

When the library receives an AW_EXIT command (indicating that the terminal's exit key or key chord has been pressed) an event of type **ExitEvent** is reported. When the event is received, the application should close down in as orderly a way as possible..

```
typedef struct {
      int                type;
} AWExitEvent;
```

AWNextEvent()

```
AW_Status AWNextEvent( AWEvent       *eventp,
                       AW_Boolean    block,
                       AW_Boolean    peek)
```

Description: This function attempts to read the next event. The type and parameters of the event will be copied into the event structure pointed to by *eventp*. This structure must have been previously allocated by the application (either statically as a variable or dynamically) before AWNextEvent() is called. If *block* is true and there are no events on the library's internal event queue then the function will block until the next event is read otherwise the routine will return without reading an event. If *peek* is true then any event read will not be removed from the internal event queue and will be returned again by the next call to AWNextEvent().

Returns: AW_OK if an event was read, or AW_ERROR if there was no event to read or a problem was detected. If an error occurred then *aw_errno* will be set to a value other than AW_NO_ERROR.

Errors: AW_COMMS_ERROR

Keyboard Control

The application may request that the pressing of certain key combinations ("chords") be reported as a special attention event rather than as keyboard characters.

AWSetAttention()

```
AW_Status AWSetAttention( int       attn_id,
                          int       key_count,
                          int       *key_list)
```

Description: This function sets an attention. *Attn_id* is an identifier for the attention which will be included in any attention events generated by the user. *Key_count* is the number of elements in *key_list*, which is a list of the key numbers which, when pressed at the same time, will generate the attention. If *key_count* is zero, the attention whose identifier is given will be cleared and the pointer *key_list* will not be dereferenced. Several standard keyboards with key numbers are illustrated in the AlphaWindow Terminal Specification. Other key number information will be published by individual vendors.

Returns: AW_OK if the window was successfully resized, or AW_ERROR if there was a problem.

Errors:

Commands: AW_SETATTN, AW_RATTN

Application Input and Output

AWlib sends AlphaWindow commands to the standard output device and reads events from the standard input device. Applications which make their own use of these devices must co-operate with the library to avoid data loss or mis-ordering of output. An application may not read directly from the standard input device - all input must be obtained via AWNextEvent(). When AWInit() is called, the library will change the device settings of standard input and output. It is the application's responsibility to ensure that these same device settings are in force before making any other calls to AWlib functions.

It is recommended that the AWOutput() routine described below is used to send application output to the terminal. An application is, however, permitted to write directly to standard output provided that all pending output is flushed prior to calling an AWlib function. Such an application will need to call AWRoute() to direct output to the appropriate VT. Note that the library will always flush its own output prior to returning.

AWOutput()

```
AW_Status AWOutput(VT_Handle  vt,
                   char           *buf,
                   int            buflen)
```

Description: This function sends application output to the given virtual terminal. *Buf* is a pointer to the start of a buffer containing the characters to output and *buflen* is the number of bytes which have been placed in the buffer.

Returns: AW_OK if the output was successfully sent, or AW_ERROR if there was a problem.

Errors:

Commands: None

AWRoute()

```
AW_Status AWRoute(VT_Handle   vt)
```

Description: This function is for use by applications which write directly to standard output. All subsequent output data will be directed to the VT specified here.

Returns: AW_OK if the output VT was successfully switched, or AW_ERROR if there was a problem.

Errors:

Commands: AW_MPI

†Flow Control

Most applications will not need to concern themselves with flow control since data transfer between host and terminal will be coordinated for them by the window manager. Note that in any case, an application should not attempt to use the credit function described here without first determining that group 4 of the AlphaWindow protocol is supported by the terminal via the AWIsCreditsSupported() function. Most window managers will not advertise themselves as supporting group 4.

The flow control system implemented by Group 4 is based on *credits*. Posession of a credit for a virtual terminal gives permission to transmit up to 32 bytes of application data for that virtual terminal. The credit system is bidirectional, so that the application must grant credits to the terminal to allow keyboard input to be received. The transmission of AlphaWindow commands is not governed by credits.

An event of type AddCreditEvent will be reported as described above when an AW_ADDCREDIT command is received from the terminal. This event may be taken as a signal that the terminal is ready to receive the indicated number of characters on the given VT.

†AWIsCreditsSupported()

```
AW_Boolean AWIsCreditsSupported()
```

Description: This function queries the terminal to find out whether it supports the use of credits for flow control. This will be true if the terminal supports group 4 of the AlphaWindow Protocol.

Returns: AW_OK if the terminal supports credits or AW_ERROR if there is no credit support or a problem was detected. If a problem was detected then aw_errno will be set to an appropriate value other than AW_NO_ERROR.

Errors: AW_NO_MEMORY

Commands: AW_DA

†AWAddCredits()

```
AW_Status AWAddCredits(VT_Handle    vt,
                       int          credits)
```

Description: This function is called to allow the terminal to send keyboard characters on the given virtual terminal. *Credits* is measured in 32 byte blocks.

Returns: AW_OK if the credits were added, or AW_ERROR if there was a problem.

Errors: AW_COMMS_ERROR

Commands: AW_ADDCREDIT

Selection Handling

An AlphaWindow terminal supports a single selection. A particular region of a virtual terminal may be notified to the terminal and the characters within that area then become the current selection value and are highlighted in some way by the terminal. An application can request the current selection value and unhighlight it by deselecting the data.

†AWSelect()
†AWDeselect()

```
AW_Status AWSelect(VT_Handle        vt,
                   Position         start_row,
                   Position         start_col,
                   Position         end_row,
                   Position         end_col,
                   Select_Mode      mode)

AW_Status AWDeselect(void)
```

Description: AWSelect() is called to select the area of the given VT between (*start_col, start_row*) and (*end_col, end_row*). The shape of the area selected will be a rectangle if *mode* is HS_RECT or contiguous lines if *mode* is HS_WRAP. The selected area will be highlighted by the terminal, perhaps using reverse video. Any previous selection is automatically cancelled by this command. Once data has been selected, it can scroll or move according to application output and the highlight moves with it. This means that selected data which is scrolled out of the VT is lost.

AWDeselect() cancels the current selection.

Returns: AW_OK if the selection was set, or AW_ERROR if there was a problem.

Errors:

Commands: AW_SELECT, AW_DESELECT

†AWGetSelection()

```
AW_Status AWDeselect(void)
```

Description: This function is called to request the characters currently highlighted as the selection. After this function is called, an event of type SelectedDataEvent containing the selected characters will be returned by a future call to AWNextEvent(). Note that it will not necessarily be the next event returned - there may be several other intervening events.

Returns: AW_OK if the command was successful, or AW_ERROR if there was a problem.

Errors:

Commands: AW_SEND, AW_DATA

Extensions

A mechanism is defined in the AlphaWindow Application Protocol to allow protocol extensions to be defined. Each extension consists of one or more commands with well known command codes. Each command takes zero or more integer parameters and a single string parameter. The meaning of these parameters is defined by the individual extension. Some commands will be defined to have a response. The reponse is simply another extension command with its own defined command code.

The extension mechanism is bidirectional - extensions may asynchronously send commands to the application. These commands will be received as extension events as described in the section on events.

AWCallExtension()

```
AW_Status AWCallExtension( int      code,
                           int      tag,
                           int      int_count,
                           int      *intparms,
                           char     *strparm)
```

Description: This function is called to issue an extension command. *Code* is the command code for the extension and *tag* is the tag number which will be placed in any reply to this command. Replies may be dealt with synchronously be calling AWWaitExtension() with the same tag value, or asynchronously by waiting for an extension event with the appropriate tag. *Int_count* is the number of integer parameters for the command, which must be stored in an array pointed to by *intparms*. The string parameter is passed as *strparm*. If a command does not require a string parameter, a null pointer should be passed.

Returns: AW_OK if the call was successful, or AW_ERROR if there was a problem.

Errors: AW_NO_EXTENSION

Commands: AW_EXTENSION, AW_SHORT_EXTENSION

AWWaitExtension()

```
AW_Status AWWaitExtension(int       command,
                          int       tag,
                          int       *int_count,
                          int       *str_count)
```

Description: To wait for a reply to an extension command, call this function. *Command* is the command code for the desired reply and *tag* is the tag value passed in the original command. The integer whose address is given by *int_count* will be set to the number of integer parameters included with the reply. Similarly, **str_count* will be set to the length of the string parameter. AWGetExtensionParms() may then be called to retrieve the actual parameter values.

Returns: AW_OK if the call was successful, or AW_ERROR if there was a problem.

Errors: AW_TIMED_OUT

Commands: AW_EXTENSION, AW_SHORT_EXTENSION

AWGetExtensionParms()

```
AW_Status AWGetExtensionParms(int         *intparms
                              char        *strparm)
```

Description: This function is called after an event of type ExtensionEvent has been reported or after a call to AWWaitExtension(), to retrieve the parameters of the extension command. The integer parameters are placed into the area of memory pointed to by *intparms*. This area must be at least large enough to hold the number of integers specified by the *int_count* member of the extension event structure. Similarly, the string parameter is copied to **strparm*, which must be a memory block large enough to hold *str_count* characters plus one character as a null terminator.

Returns: AW_OK if the call was successful, or AW_ERROR if there was a problem.

Errors:

Commands: None

Appendix A - Type Definitions

A number of new types are defined in the header file <awlib.h>. Note that types associated with events are defined in the section on event handling. A few other structure types are defined with the functions which use them.

```
typedef short AW_Status;

#define AW_OK ((AW_Status) 0)
#define AW_ERROR ((AW_Status) -1)

typedef short VT_Handle;

#define NULL_VT_HANDLE ((VT_Handle) 0)

typedef short W_Handle;

#define NULL_WIN_HANDLE ((W_Handle) 0)

typedef short Position;

typedef short Dimension;

typedef short AW_Boolean;

typedef enum {
      WT_MAIN = 1,
      WT_TRANSPARENT = 2
} Window_Type;

typedef enum {
      WS_NORMAL = 1,
      WS_MINIMISED = 2
} Window_State;

typedef enum {
      TF_NORMAL = 1,
      TF_TRANSIENT = 2
} Transient_Type;

typedef enum {
      BS_THICKNORMAL = 1,
      BS_THIN = 2,
      BS_NONE = 3,
      BS_THICKBOLD = 4,
      BS_GHOSTOUTLINE = 5
} Border_Style;

typedef enum {
      PH_NORMAL = 1,
      PH_PRIVATE = 2
} Private_Hint;
```

```
typedef enum {
      MS_BSOFT = 1,
      MS_BHARD = 2
} Bound_Type;

typedef enum {
      CT_TERMINAL = 1,
      CT_WMGR = 2
} Terminal_Type;

typedef enum {
      BD_DETACH = 1,
      BD_STRETCH_N = 2,
      BD_STRETCH_E = 3,
      BD_STRETCH_S = 4,
      BD_STRETCH_W = 5,
      BD_STRETCH_NE = 6,
      BD_STRETCH_SE = 7,
      BD_STRETCH_NW = 8,
      BD_STRETCH_SW = 9,
      BD_MOVE_ALL = 10,
      BD_SLIDE_H = 11,
      BD_SLIDE_V = 12
} Attach_Type;

typedef enum {
      SR_PROMOTE = 1,
      SR_DEMOTE = 2
} Stack_Type;

typedef enum {
      HS_RECT = 1,
      HS_WRAP = 2
} Select_Mode;
```

Appendix B - Error Types

The following error types are defined in the header file <awlib.h>:

AW_NO_ERROR
: No error has been detected.

AW_NOT_AW_TERMINAL
: The terminal does not appear to support the AlphaWindow protocol.

AW_BAD_VT
: The VT handle supplied does not refer to a VT created by this client.

AW_BAD_WINDOW
: The window handle supplied does not refer to a window opened by this client.

AW_BAD_TYPE
: The window type supplied was invalid.

AW_BAD_TRANSIENT
: The transient hint value supplied was invalid.

AW_NO_MEMORY
: The function was unable to allocate memory.

AW_TIMED_OUT
: A reply from the terminal was not received during the timeout period.

AW_COMMS_ERROR
: An error occured in reading from or writing to the window manager or terminal.

AW_BAD_BORDER
: The border style supplied was invalid.

AW_BAD_LENGTH
: The length parameter supplied was invalid.

AW_NO_MOUSE
: The terminal does not support a mouse.

AW_NO_DECORATIONS
: The terminal does not support window decorations.

AW_NO_EXTENSION
: The terminal does not support the extension command group.

AW_BAD_EMULATION

The emulation requested is not supported by the terminal.

AW_UNDEFINED_ERROR

An error not covered by the above classifications occured. It is acceptable for this error type to be generated by any function.