General Input Device Emulating Interface

(GIDEI) Standard

Version 2.2

University of Wisconsin
2107 Engineering Centers Bldg.
1550 Engineering Dr.
Madison, WI 53706

Please send any comments regarding this proposal to:

College of Information Studies
Room 4105 Hornbake Bldg, South Wing
4130 Campus Drive
College Park, MD 20742
Tel: (301) 405-2043

Forward

This document is reproduced and revised with permission. Revision 2.2 is a re-publication version 2.1 (which appears to be no longer available on the internet). This revision is intended to make the document more readable, document and correct minor errors while remaining completely true to the intent, and style of the original standard.

The GIDEI standard was originally produced as guidance to the community of developers and producers of augmentative and alternative communication [AAC] devices. The technology of these devices has moved on to USB connectivity or wireless technologies. Computer operating systems have evolved, first to include some methods described herein, and lately to omit them as part of the standard OS. An embodiment of this standard remains available from AAC INSTITUTE as AAC Keys. AAC Keys is free but unsupported software and has issues with some modern USB serial device drivers.

AAC Keys has found acceptance also by students, designers, engineers and hobbyists of imbedded hardware and software typified by the Arduino community and has relevance in 2023.

Table of Contents

GIDEI Introduction

The General Input Device Emulating Interface (GIDEI) Proposal defines a connection and data communication protocol between alternate input devices (e.g., augmentative and alternative communication [AAC] devices) and interfaces which emulate standard computer input devices.

There are a number alternate input or AAC devices capable of emulating the standard computer input devices. However, computers as they are currently designed cannot understand the output from an AAC device without some kind of communication translation taking place. This communication translation is usually accomplished by the emulating interface. The emulating interface can be a hardware device (e.g., Trace Transparent Access Module) or special software running on a computer (e.g., "SerialKeys" feature of AccessDOS) that converts the characters and commands transmitted over the serial cable from the alternate input device into keystroke and mouse actions understood by the receiving computer. In either case, a hardware interface (e.g. a serial port) is necessary to supply or support the connection from the alternate input device to the emulating interface. The hardware interface (e.g., a serial port) can be physically located on an external hardware device attached to the computer or be part of the computer itself.

As you read or refer to this GIDEI Proposal, you will need to distinguish between what the GIDEI Proposal is defining (e.g., the connection and data communication protocol) and what other pieces are necessary to allow alternative input or AAC devices to function as computer input devices (e.g., keyboards, mice, etc.).

For a quick reference, several of the GIDEI commands which comprise the "data communication protocol" are listed in the section titled "GIDEI Commands At A Glance." Information concerning the GIDEI "connection protocol" requirements are covered later in this document in the section titled "Hardware Connection."

In addition, Appendix A has added information on an industry wide effort that promises to make PC installation and use easier for all computer users. Collectively known as the External COM device "Plug- and-Play" draft specification, it is an attempt to simplify the installation, device identification, and initial communication for devices attaching on the "serial" port, which may have a far reaching effort towards "ease-of-use" for AAC devices accessing a computer via GIDEI.

GIDEI Purpose

The main purpose of the GIDEI Proposal is to provide developers of AAC and other alternate input devices with a viable approach that enables them to use their devices as replacements for the normal keyboard and mouse on general purpose computers. This allows people who are "unable" to effectively operate the normal keyboard and mouse, to access computer systems and software from their alternate input device.

Keep in mind that the General Input Device Emulating Interface (GIDEI) Proposal defines a connection and data communication protocol only, between the alternate input devices and other emulating interfaces which emulate standard computer input devices, not to any software or hardware designed to do translation.

Background

A person who is unable to access a computer is at a great disadvantage in today's society. Many schools and jobs require the use of computers daily. The person who is physically unable to use a computer keyboard or mouse can not participate in certain activities on an equal level with others.

Technology has evolved to the point where electronic aids are available for many people with physical disabilities to assist them in doing many of the same things that others are doing on standard computer systems. For example, several aids can perform word processing features. Yet there still exists the need to access the same computers and software that are used by others. In schools and in the workplace, an individual is often required to use the same software programs that others are using.

Fortunately, many special electronic aids used by people with disabilities can be used as alternate input devices for standard computers. Any device which is capable of connecting to a computer as described in this GIDEI Proposal and also capable of transmitting ASCII information arranged to conform with the GIDEI data communication protocol, should be capable of acting as an alternate input device.

The following figures should help you to better understand the terminologies GIDEI, alternate input device, and emulating interface. Each figure contains a picture of a computer, showing the connection for the standard keyboard and mouse input devices. Each figure also contains a picture of an alternative input device (e.g., AAC) device that is also connected in some fashion to the computer. It is this AAC device-to- computer connection that is the critical component you need to understand, when determining if you need-to- follow the GIDEI Proposal for connection and data communication purposes between your AAC device and the computer.

The detailed discussions which follow are designed to assist you in better understanding the proper application of the GIDEI Proposal, with or without the figures as a reference.

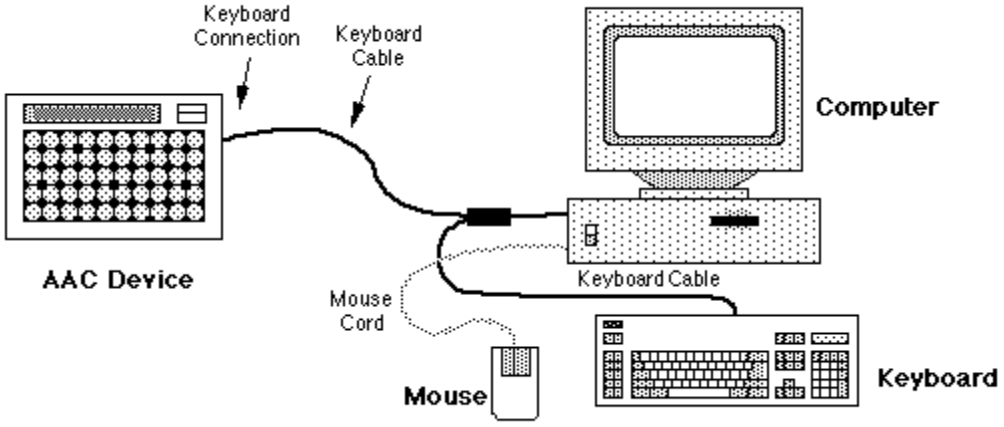

Figure 1

The example in Figure 1 shows an AAC device connected directly to the computer using a serial cable. The serial cable from the AAC device connects directly to the computer serial or "com" port, which is usually located at the rear of the computer. The user in Figure 1 wishes to make selections on their AAC device, transmit the information across the serial cable to the computer, and expects the computer to understand that the information represents keyboard or mouse actions. In Figure 1, the "connection" protocol as defined in the GIDEI Proposal includes the serial cable, its internal wiring or "pin-out" arrangement, and the amount, speed, and characteristics of the information the user sends from their AAC device to the computer. The "data communication" protocol as defined in the GIDEI Proposal concerns the contents of the information or "data packets" themselves, that the user either creates or which may be pre-stored on their AAC device.

You might expect that once the user sets up their AAC device to conform to both the connection and data communication protocol as defined in the GIDEI Proposal, that their AAC device would be able to communicate with and control the computer. Unfortunately, without one additional missing piece, the computer will not be able to understand what the AAC device is transmitting.

Computers are very powerful tools. However, from a user input standpoint, most computers only understand the user when given input or instructions from the keyboard or the mouse. By design, keyboards and mice each use their own language to communicate with the computer. Therefore, once the AAC device is connected to the computer as shown in Figure 1 (e.g., either directly through the use of a serial cable or indirectly via some kind of wireless link), something must act as a "go-between" to translate the AAC device language into the keyboard or mouse language understood by the computer. Since the information transmitted from the AAC device is well defined by the GIDEI Proposal data communication protocol, a good portion of the translation work is already completed. In Figure 1, the missing "go-between" piece is the software that functions as the "emulating interface" on the computer receiving the alternate input device information. Emulating interface software would do the translation of the information sent from the AAC device into keyboard and mouse language understood by the computer. The emulation interface software is the missing "go- between" piece that allows the AAC device which is connected directly to the computer com port to "emulate" these standard input devices. The "SerialKeys" feature provided as part of the AccessDOS software package is an example of "emulating interface" software.

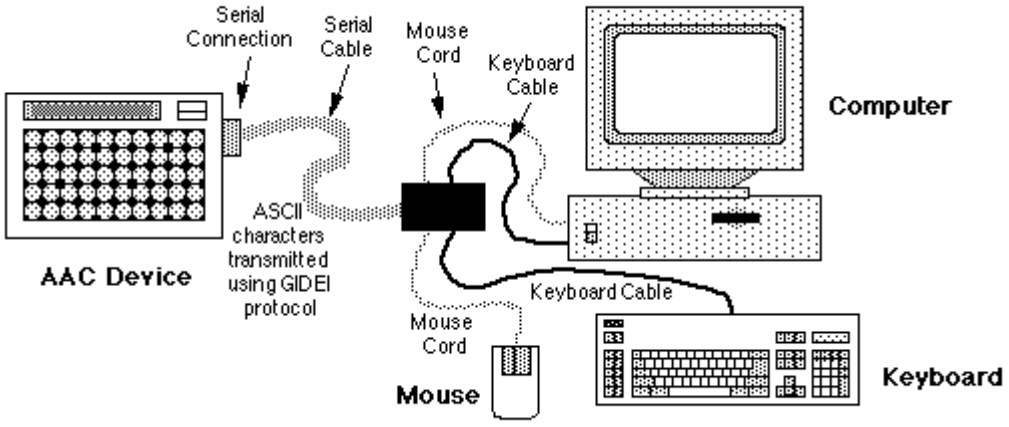

Figure 2

The AAC to computer connections shown in Figure 2 are very similar to the example shown in Figure 1, except the serial cable coming from the AAC device is now connected to a "black box" outside the computer rather than directly to the computer. (Again, please note that the serial cable could also be a wireless serial link). The black box has cables coming "from" the keyboard and mouse, and cables "exiting" the black box which connect to the computer at the same place the keyboard and mouse cables would have been plugged into the computer, had the black box not been used. The black box also has a connection for the serial cable used to connect to the AAC device. Again the "connection" protocol as defined in the GIDEI Proposal includes the serial cable, its internal wiring or "pin- out" arrangement, and the amount, speed, and characteristics of the information the user sends from their AAC device to the black box. The "data communication" protocol as defined in the GIDEI Proposal concerns the contents of the information or "data packets" themselves, that the user either creates or which may be prestored on their AAC device.

In this example, information is sent from the AAC device to the black box. Therefore, the black box needs to perform the necessary translation of the AAC device information prior to sending it to the computer through the keyboard and mouse cables. The black box becomes the missing "go-between" piece in this example. Since the black box is connected between the standard input devices and the computer, it understands the language used by the keyboard and mouse when talking with the computer, and is therefore capable of allowing the AAC device to again "emulate" the standard input devices. In this case, we would refer to the black box as an "emulating interface device," since it is actually a piece of hardware which a user could take from computer-to-computer, as their needs might dictate. The Trace Transparent Access Module (T-TAM) is one example of an "emulating interface device."

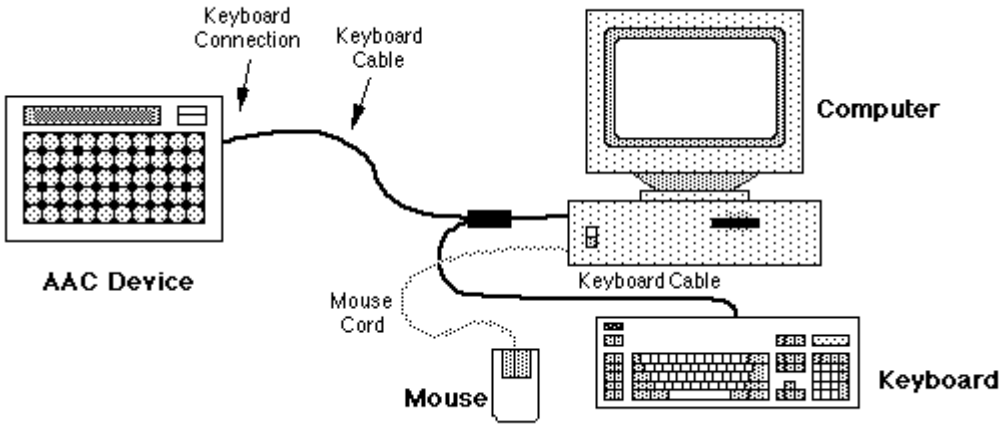

Figure 3

Figure 3 shows one more example connection. However, the example in Figure 3 shows how a non-GIDEI alternate input device might connect to a computer. The AAC device in Figure 3 is connected directly to the computer using a keyboard cable. Since there is no serial cable or wireless serial link involved, this example does not use the GIDEI Proposal.

The AAC device in Figure 3 does not use any kind of GIDEI connection, because this AAC device already has a keyboard emulation interface built into it which understands the keyboard language of the computer it is connected to. In this example, we would say that the AAC device is acting as both the "alternative input device" and the "emulating interface device." Some individuals may also call an AAC device with "direct keyboard connection" capability a

"keyboard emulating interface," since the AAC device connects directly to a keyboard cable and "emulates" or looks like another keyboard to the computer. This terminology can be confusing however, since it does not distinguish between a device which is capable of connecting directly to the keyboard port or a device which emulates the keyboard using an external emulator. Perhaps a preferred term for this type of device would be to describe it as an alternative input device with built-in keyboard emulation. However, as stated above, this combination is still another example of keyboard emulation.

As technology advances, we expect more and more AAC devices to be designed capable of connecting directly to the computer as substitute or additional keyboards, mice, etc. These devices would not need to use the GIDEI connection and data communication protocol. However, until all alternative input devices are so equipped, the GIDEI Proposal provides a consistent method for users to connect with and operate a computer.

Standardization

The reasons behind the effort to standardize the GIDEI is based on several important and practical concerns:

> Often, different emulating interfaces must be designed for each model of computer, keyboard and mouse, and these interfaces may be designed by different developers. Standardizing the language or "interface" part of the emulating interface will make emulating interfaces from different developers compatible. Alternate input devices using the GIDEI connection and data communication protocol will find a greater number of computers that are accessible. This increases its usefulness for users while at the same time expands the market for manufacturers of alternate input devices.
>
> Many people need access to two or more different types of computers. If the only way they can obtain that access is through a GIDEI connection, it would be beneficial to the user if the GIDEI worked the same (at the user level) for all computers, keyboards and pointing devices. There is little enthusiasm on the part of users, clinicians and developers to have to deal with multiple GIDEI formats and protocols.
>
> For schools and clinics it is important that a emulating interface on a computer be compatible with a variety of alternate input devices so that different people can access or share the computer. For public computer systems, such as computer based card catalogues in public libraries, it is important that the computer be accessible to everyone. Having a single GIDEI connection and data communication protocol that all alternate input devices can implement will provide that access in the most advantageous solution.

In short, a standard as proposed by the GIDEI connection and data communication protocol simplifies development, makes it easier for people to connect their augmentative or alternate communication devices as alternate computer input devices, and provides access to more computers for those people that are unable to use the normal keyboard and mouse effectively.

GIDEI Overview

The GIDEI Proposal defines the connection and data communication protocol between alternate input devices and an emulating interface. The alternate input device, using this protocol, transmits information to the computer. The emulating interface then processes that information to determine which keystrokes to type or what mouse actions to perform on the

computer. If it is a hardware emulating interface device, it may need to mimic the electrical signals that the normal keyboard and mouse exhibit. If it is a software emulating interface, it will place the keystroke and mouse information into locations within the computer or its operating system, where application programs will think it came from a user typing on the keyboard or from a user operating the mouse.

Some confusion arises when software or hardware developers state that they have a "general input device emulating interface," either in software or hardware. What they are referring to usually, is the fact that their interface can handle "general" input, thus it isn't limited to translating alternative input device information into only keyboard or only mouse languages. While reading this document, the General Input Device Emulating Interface (GIDEI) refers only to the connection and data communication protocol, not to any software or hardware designed to do translation.

The GIDEI connection protocol is designed to allow both wired and wireless links for GIDEI communication. Specifications have been made to allow for some degree of flexibility while taking into account the limitations of most current alternate input devices. Basic operation of the GIDEI is possible with virtually any alternate input device with a serial port and advanced options are included for alternate input devices that have additional capabilities.

The GIDEI Proposal defines two modes of operation as part of the data communication protocol for sending keystroke and mouse information: Character Mode and Escape Sequence Mode. In Character Mode, the emulating interface would understand to convert ASCII characters received from the alternate input device directly into whatever keystrokes are necessary to produce those same characters on the computer. The Escape Sequence Mode is designed to provide a way for a person using an alternate input device to type keys that have no ASCII equivalent (e.g., Home, PageUp, etc) as well as to allow the user to "hold" two or more keys down at the same time, as is required by many computers (e.g., Ctrl-Alt-Del). Escape Sequence Mode also allows the user to specify mouse actions such as clicking, double clicking and moving. Finally, Escape Sequence Mode is used to change certain operational features the GIDEI Proposal supports.

The GIDEI Proposal was developed with thought given to both hardware and software emulation interface implementations. Features such as key repeat and adjustable typing rate were deliberately removed from this proposal to take into account the fact that some emulating interface implementations could not guarantee access to a timing mechanism. Memory requirements were also considered when Commands and Key Names were established as part of the GIDEI data communication protocol. While the length of some of the Key Names could be shortened, this would have made it much more difficult for users to understand. Since many of the alternate input devices today require the user to program these escape sequence strings in "squares" or "selections" on their alternate input device, user readability was given greater priority than device memory requirements.

Previous versions of the GIDEI were concerned with maintaining a degree of compatibility with previous versions of the Keyboard Emulating Interface (KEI v1.0) standards. However, to provide better support for the newer and more powerful communication aids, GIDEI2 no longer guarantees such "backward" compatibility.

GIDEI Commands At A Glance

Keyboard Commands

Typing Characters: simply send the character, which is sometimes referred to as an "ASCII character" to the emulating interface. Some examples of ASCII characters are the lowercase letters "a-z", uppercase letters "A- Z", most common punctuation characters like comma ",", period ".", etc., and the numbers "0 through 9."

Typing Specific Keys: each key is designated by a name. To type the key, use the Key Name in an escape sequence. An escape sequence is simply the ESCAPE character (ASCII 27, represented as "<esc>" below), followed by a sequence of other ASCII characters, and ending with the PERIOD character (ASCII 46,"."). (Note: Some Many AAC devices can generate an ESCAPE character using a combination of the control key plus a left bracket (e.g., control+[), which may appear on the AAC device display as "^[" or as a blank character. Others have the ESCAPE character programmed on a keyboard layout page.)

<esc> Key Name .
For example: <esc> pageup .

Other Key Actions: to perform other actions, use escape sequences of the general form:

<esc> , command [, Key Name 1] [, Key Name 2] [, Key Name 3] [, Key Name 4] [, Key Name 5] .

Valid keyboard commands are:

combine
used to type one to five key(s) "simultaneously."

<esc> , combine , ctrl , alt , del .

hold
holds the specified key(s) down until the next key is "typed."

<esc> , hold , shift .

lock
locks key(s) down until the rel command is used to release them (e.g., a "key down" is transmitted with no "key up").

<esc> , lock , ctrl .

rel
releases all or only specified "locked" key(s) (e.g., sends "key up") and clears any pending hold keys.

<esc> , rel .

Mouse Commands

Mouse button actions are performed using escape sequences of the general form:

<esc> , command , button identifier .

Button identifier now refers to a mouse button action, such as pressing mouse button #1, rather than a location, such as pressing the "left" mouse button. This distinction is being made because many operating systems now allow the user to reprogram the buttons on the mouse to perform any desired action. Therefore, users are encouraged to follow the new button identifiers, rather than previous GIDEI conventions which defined a "left" or "right" mouse button. If the button identifier is not included, typically the "default" button is activated with the mouse command.

Valid mouse commands are:

click
sends a "button down" immediately followed by a "button up" of the specified mouse button identifier(s).

<esc> , click .
- or -
<esc> , click , but1 .

dblclick
sends two click actions of the specified mouse button identifier(s).

<esc> , dblclick .

moulock
locks the specified mouse button identifier(s) down (e.g., a "button down" is transmitted with no "button up," and could be more than one button at a time.)

<esc> , moulock , but1 .

mourel
Releases "locked" mouse button identifier(s) (e.g., sends "button up").

<esc> , mourel .

Mouse movement actions are performed using the escape sequence formats shown below:
Anchor, used to remember a specific location of the mouse cursor for subsequent return actions.

<esc> , anchor .

(The next single lowercase letter "a" through "z" following an "anchor" command becomes the "name" of the current mouse cursor screen position, thus allowing 26 separate mouse cursor locations to be "anchored" or saved.)

move
Moves the mouse cursor a specified distance in the specified direction(s).

<esc> , move , 1 X direction , 1 Y direction .

mougo
Moves the mouse cursor in a continuous motion, in the direction specified and also at the speed specified. On systems which support this type of mouse motion using an alternate method (e.g., MouseKeys), this command is not necessary.

<esc> , mougo , downleft , 5 .

goto
The "goto" command works in two different fashions, depending upon whether an absolute location or an "anchor" location (see "anchor," above) is given.

A) It moves the mouse cursor to a specified absolute screen location when both an X and Y position is indicated.

<esc> , goto , X position , Y position .

B) It moves the mouse cursor to a specified screen location, based upon that location having been preassigned using the "anchor" command (see above).

<esc> , goto .

(The next single lowercase letter "a" through "z" following this "goto" command is the "name" of the mouse cursor screen position "anchor" the user wishes to return to.)

moustop
allows the user to halt the mouse cursor after the mouse cursor was put in continuous motion using the mouse "mougo" command (see above). On systems which support this type of mouse motion using an alternate method (e.g., MouseKeys), this command is not necessary.

<esc> , moustop .

moureset
initializes the mouse settings and coordinates to (0,0) and moves the pointer to the upper left hand corner of the screen.

<esc> , moureset .

Miscellaneous Command(s)

Initialization and configuration of the GIDEI can be performed with the following procedures and commands.

If a user attaches to a computer which has the "emulating interface" already running, and which conforms to the GIDEI Proposal, they need to first establish a connection or get their AAC device to send characters at the same "rate" at which the emulating interface is accepting characters. The easiest and quickest way to ensure this, is to cause the emulating interface to reset to a "known" state. To accomplish this, if the user can cause three "framing" communication errors to occur, any emulating interface which follows the GIDEI Proposal is supposed to perform a reset. If the user transmits three consecutive NULL characters (ASCII 0) at 300 baud, any GIDEI compliant emulating interface, if not already at 300 baud, should reset itself to 300 baud. This ensures a "known-data- rate" communication connection between the user's AAC device and the emulating interface. (See the section on Hardware Connection for a more in-depth discussion of this "Initial Connection Protocol.")

baudrate

sets the baud rate of the GIDEI.

<esc> , baudrate , XXXXX .

When the AAC user is communicating with the emulating interface at a known rate, the baudrate command can be used to change to another rate, should the user wish to do so.

Definition of Terms

Alternate input device: Any device that can electronically output ASCII characters and is used as an alternative to the regular keyboard or other input device on a general purpose computer system. An alternate input device is used because a person finds it more accessible and effective than the regular keyboard (or other input device).

ASCII: ASCII is an acronym which stands for the American Standard Code for Information Interchange. What this means to the computer user is that usually numeric values represent characters inside the computer. For example, the ASCII code for the uppercase "A" is "65." Table 1 provides a listing of ASCII characters and their corresponding numeric or decimal code.

Emulating Interface: A "system" that accepts information from an alternate input device and transforms it into keystrokes or other input information in the computer.

Escape Character: The single character used to indicate the beginning of a GIDEI Escape Sequence. It is ASCII 27. In this document the escape character is designated by "<esc>."

Escape Sequence: A character string that starts with an escape character (ASCII 27) and ends with a period character (ASCII 46). Escape Sequences are used to send special commands and keystrokes.

Hardware emulating interface: Usually an "external" device that attaches to the computer and provides connection "ports" for AAC devices while simultaneously interacting with the computer in such a way that the computer is unaware that any external device is attached.

Key Name: The ASCII string of characters assigned by the GIDEI Proposal to designate each key on the normal keyboard.

Neutral Keyboard: The state of the keyboard when all "toggling" keys are in their "untoggled" position.

Secondary Key Function: One or more actions produced by a key when it is typed in combination with other keys. Secondary Key Functions are normally activated by holding down, for example, a [Shift] key, an [Alt] key, an [Option] key, a [Ctrl] key, etc., along with the key.

Software emulating interface: Software which is implemented as a co-process within the target computer. Typically, it is a program resident in the computer which uses the computer's serial port to communicate with the alternate input device. It is activated when the serial port receives characters. Once activated, it interprets the keystroke and mouse information and makes these keystrokes available to the operating system in that computer.

Special Keys: Keys that do not produce ASCII characters. An example is the "right arrow" key.

Transparent Interface: An interface that is invisible to or undetectable by the computer system and the software running on it. An emulating interface whose implementation is not distinguishable from the way the normal keyboard works. In other words, it is an interface that

operates with a computer and the software running on it exactly the same way as the normal keyboard would operate. Ideally, all interfaces should be transparent to insure access to the same programs that can be operated from the normal keyboard or mouse.

XOFF Character: The character sent by the GIDEI to the alternate input device to tell it to stop transmission. The XOFF character is ASCII 19 (control-S).

XON Character: The character sent by the GIDEI to the alternate input device to tell it to resume transmission. The XON character is a ASCII 17 (control-Q).

Hardware Connection

This section defines specifications for the hardware connection or hardware interface portion of the system.

The hardware interface is the component of the system which the user's AAC device "directly connects to." It handles the physical connection, signal interfacing, communications protocol and data buffering.

The GIDEI Proposal data connection protocol uses the RS-232C communication link for transferring information from an external AAC device to the hardware interface. The hardware interface itself might be under the control of emulating interface "software" running on a computer or a separate external "black box" connected to the computer.

See also Appendix A, regarding "Plug-and-Play."

Serial Connection

The serial connection is based on EIA Standard RS-232-C: Interface Between Data Terminal Equipment and Data Communication Equipment Employing Serial Binary Data Interchange, August 1969, Electronic Industries Association, Washington, D. C. 20006.

Of the 25 lines defined, four are used for the GIDEI Proposal.

Transmitted Data Line
The Transmitted Data line is EIA Interchange Circuit BA. It is used to send information from the external device to the hardware interface.

The hardware interface plays the role of the DCE (data communication equipment). The alternate input device or "external device" would connect to the hardware interface with a serial cable or wireless serial link, as discussed earlier in Figures 1 and 2. The hardware interface is usually the computer running emulating interface software or a black box connected to the computer.

Received Data Line
The Received Data line is EIA Interchange Circuit BB. It is used to send information from the hardware interface to the external device.
Clear to Send Line
The Clear To Send (CTS) line is EIA Interchange Circuit CB. This is the hardware handshaking line controlled by the hardware interface.
Signal Ground Line

The Signal Ground or Common Return line is EIA Interchange Circuit AB. It is used to establish a common voltage reference between the connected devices.

Connector

The connector mounted on or associated with the hardware interface may be either a female 25 pin D or female 9 pin D connector. The AAC user should examine the computer serial port connectors to determine the necessary requirements for each system they intend to connect to.

Pin Assignments (for the AAC, or DTE, device)

(assuming the AAC device has a 25 pin D connector)

Pin 2 is Transmitted Data (EIA Interchange Circuit BA).
(Direction of data flow from the external AAC device to the hardware interface)
Pin 3 is Received Data (EIA Interchange Circuit BB).
(Direction of data flow from the hardware interface to the external AAC device)
Pin 5 is Clear To Send (EIA Interchange Circuit CB).
(Direction of data flow controlled by the hardware interface, which can be a hardware "black box" device or emulation interface software)
Pin 7 is Signal Ground (EIA Interchange Circuit AB).

(Assuming the AAC device has a 9 pin D connector)

Pin 2 is Received Data
Pin 3 is Transmitted Data
Pin 5 is Signal Ground
Pin 8 is Clear To Send

Transmission Format

Each character transmitted from the external AAC device and the hardware interface will use RS-232C voltage and logic levels. Transmissions follow standard asynchronous communications protocol with the following format:

1 Start Bit
8 Data Bits:
0 Parity Bits
1 Stop Bit

Transmission Rate

The hardware interface is configured to initially run at 300 baud. (Note: the hardware interface may provide the capability to initially configure at a baudrate other than 300 baud. This capability is a function of the hardware interface, and is not part of the GIDEI Proposal. The GIDEI Proposal does however, define an "Initial Connection Protocol to handle situations where the hardware interface does not initially configure itself at 300 baud.)

The user may change the baud rate via an escape sequence command sent to the hardware interface. Valid baud rates are 300, 1200, 2400, 4800, 9600, and 19200.

Handshaking

In order to control the flow of characters being sent to the hardware interface, both hardware and software handshaking methods are employed. Either interface will simultaneously support both methods thereby allowing the external AAC device the option of selecting the method most appropriate for its design.

The descriptions below pertain to both devices during normal operation after transmission has been established. There are some changes that must be noted when the devices are initially connected, or when a baud rate change takes place. These two cases are explained in separate sections later in this document.

With hardware handshaking, the hardware interface controls the signal on the Clear To Send line. It asserts this line when it can receive data and lowers it when it is not ready to receive more data.

The external AAC device, if using this handshaking method, must check the status of this line before each character is transmitted. It may only send data if the line is asserted.

Software handshaking is a system where the hardware interface, using ASCII characters, transmits its status information to the external AAC device on the Received Data line. The hardware interface sends an XON character (ASCII 17) to signal that it is ready to receive data and sends an XOFF character (ASCII 19) to signal that it is not ready.

The external AAC device, if using this handshaking method, must check its receive buffer before each transmission to determine if an XOFF has been sent. If so, it must wait until an XON is sent before sending any more data.

Hardware Interface Handshaking

When the hardware interface is ready to receive characters from the external AAC device, it places the CTS line into the ON state (logic 0, +3 to +15 volts) and sends one (1) XON character at the current baud rate to the external device over the Received Data line.

To suspend the transmission of characters by the external (AAC) device, the hardware interface puts the CTS line into the OFF state (logic 1, -3 to -15 volts) and sends one (1) XOFF character at the current baud rate to the external device over the Received Data line.

If the hardware interface lowers the CTS line and sends an XOFF while a character is being transmitted by the external device, transmission will continue until the current character has finished being transmitted.

It may turn out that the external AAC device, due to the buffering capabilities of its serial hardware, may not be able to stop the transmission of characters immediately. In this case, one or more characters might still be sent after the hardware interface has signaled it to stop. To accommodate this, the hardware interface will be required to be able to receive four (4) more characters after the initial XOFF has been sent to the external device. In addition, after each of these "extra" characters is received, the hardware interface will send another XOFF back to the external device.

External Device Handshaking

The external device must use at least one of the handshaking methods supplied by the hardware interface: either the hardware method or the software method. It should check the status of the CTS line or check its own receive buffer for an

XOFF character before it transmits each character. If either check indicates that the hardware interface is not ready, the external device should not send any more characters to its transmitting circuitry.

Once the external device has received the XOFF character or has detected the lowering of the CTS line, it must wait for an XON or an asserted CTS before any additional data is sent to the hardware interface.

There are a few exceptions to this requirement. See later.

If the external device is using the XON/XOFF handshaking method, it is recommended that the external device periodically interrogate the status of the hardware interface while it is waiting for an XON. (See next section)

There is no timing specification given to define what "periodically" means. The intention is that the external device checks at reasonable intervals. Not so often that the hardware interface spends all its time handling the serial transmission and therefore can't get anything else accomplished, but often enough to avoid an unreasonable delay for the user.

Status Inquiry

There exists a way for the external device to interrogate the status of the hardware interface to guard against the case where the external device using software handshaking "missed" the transmitted XON character.

To do this, the external device is allowed to send a NULL character (ASCII 0) at the current baud rate to the hardware interface. If the hardware interface is ready to receive keystroke data it will receive this character and then send an XON back to the external device. While the hardware interface must monitor the character data for this character, it is still passed on to its input buffer just like any other character.

If the hardware interface is not ready to receive keystroke data, it will send nothing back.

No character is sent back in this case (instead of sending an XOFF) to guard against potential problems. First, if the external device missed the original XON (which is why it is interrogating), chances are that it could miss an XOFF when the hardware interface is interrogated. In that case the external device is hung up waiting for the XOFF response. Secondly, the hardware interface may have sent the original XOFF because it was too busy to handle any more serial data for a while. If this is the case, the NULL character would not be received or interpreted by the hardware interface so it couldn't possibly send the additional XOFF.

So no matter what the reason is that the hardware interface is busy, the end result, as seen by the external device, is that the hardware interface did not send back an XON. The external device should try again later.

This interrogation method is useful (and recommended) for external devices using the software handshaking system to insure that transmission does not hang up because an XON character was missed.

Handling Framing Errors

Characters received with framing errors, or any other transmission errors will not be passed on to its input buffer. However, the number of consecutive characters received with framing errors is remembered.

If the hardware interface receives three (3) consecutive characters with a framing error, it will:

1. Send an XOFF at its current baud rate and lower the CTS line

2. Change to 300 baud (if not at this rate already)

3. Raise the CTS line and send out an XON at 300 baud after the above steps are completed in order.

Besides the benefit of setting the hardware interface to a known state, this requirement will also help to insure the quality of the transmission by not allowing baud rates which produce repeated erroneous characters.

Initial Connection Protocol

There is no sure way to determine what state the hardware interface is in when first connecting to it. To overcome this problem, an Initial Connection Protocol is specified which, when followed, will configure the hardware interface into a default state (initialized state) where communication may begin. The default state is for the hardware interface to be operating at 300 baud.

The hardware interface may have been left at a different baud rate by the previous user or it may be in the middle of an escape sequence. Even if it was just installed, noise on the lines when the external device is connected could possibly put the hardware interface into an escape sequence. (This probably pertains more to wireless link serial systems.)

To begin the initial connection, the external device must be configured to a default state also.

External Device Configuration

The external AAC device must be set to 300 baud.

If software handshaking is being used, the external device should assume that the hardware interface is ready to receive characters. *The hardware interface is assumed to be installed and ready. Certainly, this should be the case if it has been idle for a short amount of time or just installed/powered up.*

Until the hardware interface has changed to 300 baud, the external device should ignore any characters received from the hardware interface.

The hardware interface may be sending information back to the external device but you have no way of knowing the baud rate at which it is transmitting until it is initialized.

Once the external device is set up, it sends three NULL characters (ASCII 0) at 300 baud to the hardware interface. This causes one of two things to happen:

Sending NULL characters at 300 baud is the recommended way to insure that three framing errors are caused.

In addition, using the NULL character will avoid the possibility of one type of error that can occur if the hardware interface is at a higher baud rate. What can happen is that the hardware interface could mistakenly assemble a character (or characters) if the data bits are not all logic 0 because of the unsynchronized bit transmissions.

> If the hardware interface is at a baud rate other than 300, the NULL characters will cause framing errors in the hardware interface. This will cause the hardware interface to switch to 300 baud.
>
> If the hardware interface was already at 300 baud, the NULL characters do nothing other than cause XONs to be sent back to the external device because of the Status Inquiry. These XONs are ignored by the external device. While the NULL characters are passed on to its input buffer, it is unlikely that it would cause any problems for any connected modules.

Regardless of what baud rate the hardware interface was in, it is now at 300 baud and the external device has established communication with the hardware interface.

To complete the initialization, the external device then sends an Escape Sequence Command to the hardware interface to cause it to reset to Character Mode.

> Remember, handshaking is now in affect so the external device must either check the CTS line, or wait for an XON from the hardware interface at 300 baud. The external device using software handshaking should interrogate the hardware interface to make sure that it is ready. There may be a delay between the time the hardware interface gets the last framing error and when it has changed baud rate and is ready to receive characters.

The Escape Sequence Command that must then be sent is an Escape character followed by a period:

> <esc> .

(See section on Escape Sequence Mode for information about Escape Sequences.)

After receiving this command, the hardware interface will be configured in its default state, or ready to receive ASCII characters (See next section on "Character Mode" transmission.).

Reset Signal to the User

Although it is not required, it is recommended that the hardware interface give the user an indication whenever it has initialized.

Two methods are suggested:

Audible Signal

After the hardware interface resets it generates an audible signal. The signal sound should be a simple tone lasting approximately 1.0 seconds and audible in the presence of normal ambient sound by a person with normal hearing up to four feet from the computer screen.

Visual Signal

After the hardware interface resets it sends a visual signal to the user in the form of a flashing amber LED or if possible, a warning message on the Computer system's screen. The LED should be amber in color, clearly visible to a person looking at the computer screen, and should light for 10 seconds. The warning message to the computer system's screen could indicate that the hardware interface has been reset to 300 baud. For example, it could read: "SerialKeys has reset the com port to 300 baud and is now ready to continue receiving characters."

Handshaking After Changing Baud Rate By Command

The hardware interface will have the capability to change to a user specified baud rate through an Escape Sequence Command. Whenever the baud rate is to be changed, the hardware interface will carry out the following handshaking methods.

When the hardware interface is ready to change baud rate, it will lower the CTS line and send an XOFF character to the external device. If additional characters are received by the hardware interface after this point, they are ignored.

The hardware interface will then perform its baud change to the specified baud rate. After changing the baud rate, it will clear out any characters it may have received since the baud rate change began. When completed, it will signal the external device that it is ready by raising the CTS line and sending an XON character at the new baud rate.

Since the external device might not change baud rate as fast as the hardware interface, it might not properly receive the XON character. Because of this, it should employ the Status Inquiry method, if it is using XON/XOFF handshaking.

Character Mode

Character Mode is the most common mode in which an alternative input device using GIDEI data communication protocol will be running. In this mode, ASCII characters sent from the alternate input device to the computer will be converted to the keystrokes necessary to produce that ASCII character on the computer.

Table 1 shows all the ASCII characters from 0 through 127. Characters which exist above 127 (e.g., ASCII 128 through ASCII 255), sometimes referred to as "Extended ASCII," may differ due to the computer or receiving device's operating system.

If your alternate input device is capable of sending the ASCII value for the character you need, most hardware interfaces will understand how to generate the character or key for your request. If some of the keys on a particular keyboard have single character labels which are ASCII characters in the Extended ASCII 128 to ASCII 255 range, GIDEI2 supports transmission of these characters and hopefully the hardware interface will understand how to "create" the character or keyboard "keys" via emulation on the receiving computer. For example, if you require an "e" with a circumflex accent "^" on a computer in the United States, GIDEI2 supports transmission of the Extended ASCII character "234"(ISO 8859-1 Standard) direct from the alternate input device to the hardware interface. (Note: remember the hardware interface may be either an external "black box" device connected to the receiving computer or emulation interface software running directly on the receiving computer.)

Two ASCII characters are reserved by the GIDEI for special use. They are the NULL character (ASCII 0), and the ESCAPE character (ASCII 27). See the Escape Sequence Mode section to find how to type these characters.

Based on the GIDEI Proposal, an emulating interface will leave Character Mode and enter Escape Sequence Mode upon receiving an Escape character. (see the next section for information on Escape Sequence Mode) Based on the GIDEI Proposal, after start up, after an escape sequence, and after an error in an escape sequence, the emulating interface will return to Character mode.

Example: if an ASCII d is sent, the interface will type a d on the computer. If an ASCII G is sent, the interface will type a shift key and a g to produce a capital G on the computer.

The actual keystrokes that the interface will type to get the character is dependent on the keyboard that the interface is emulating. The intention of the GIDEI is to allow the external AAC device to send characters that the hardware interface then translates into the appropriate key presses or mouse actions.

Table 1: ASCII Characters With Their Corresponding Decimal Code

Character	Code	Character	Code	Character	Code	Character	Code
NULL	0	SPACE	32	@	64	`	96
SOH (ctrl-A)	1	!	33	A	65	a	97
STX (ctrl-B)	2	"	34	B	66	b	98
ETX (ctrl-C)	3	#	35	C	67	c	99
EOT (ctrl-D)	4	$	36	D	68	d	100
ENQ (ctrl-E)	5	%	37	E	69	e	101
ACK (ctrl-F)	6	&	38	F	70	f	102
BEL (ctrl-G)	7	'	39	G	71	g	103
BS (ctrl-H)	8	(	40	H	72	h	104
HT (ctrl-I)	9	)	41	I	73	i	105
LF (ctrl-J)	10	*	42	J	74	j	106
VT (ctrl-K)	11	+	43	K	75	k	107
FF (ctrl-L)	12	,	44	L	76	l	108
CR (ctrl-M)	13	-	45	M	77	m	109
SO (ctrl-N)	14	.	46	N	78	n	110
SI (ctrl-O)	15	/	47	O	79	o	111
DLE (ctrl-P)	16	0	48	P	80	p	112
DC1 (ctrl-Q)	17	1	49	Q	81	q	113
DC2 (ctrl-R)	18	2	50	R	82	r	114
DC3 (ctrl-S)	19	3	51	S	83	s	115
DC4 (ctrl-T)	20	4	52	T	84	t	116
NAK (ctrl-U)	21	5	53	U	85	u	117
SYN (ctrl-V)	22	6	54	V	86	v	118
ETB (ctrl-W)	23	7	55	W	87	w	119
CAN (ctrl-X)	24	8	56	X	88	x	120
EM (ctrl-Y)	25	9	57	Y	89	y	121
SUB (ctrl-Z)	26	:	58	Z	90	z	122
ESC	27	;	59	[	91	{	123
FS (ctrl-\)	28	<	60	\	92	|	124
GS (ctrl-])	29	=	61	]	93	}	125
RS (ctrl-^)	30	>	62	^	94	~	126
US (ctrl-_)	31	?	63	_	95	DEL	127

List 1: ASCII Characters With Their Corresponding Decimal Code

Character, Code

NULL 0 SOH (ctrl-A) 1 STX (ctrl-B) 2 ETX (ctrl-C) 3 EOT (ctrl-D) 4 ENQ (ctrl-E) 5 ACK (ctrl-F) 6 BEL (ctrl-G) 7 BS (ctrl-H) 8 HT (ctrl-I) 9 LF (ctrl-J) 10 VT (ctrl-K) 11 FF (ctrl-L) 12 CR (ctrl-M) 13 SO (ctrl-N) 14 SI (ctrl-O) 15 DLE (ctrl-P) 16 DC1 (ctrl-Q) 17 DC2 (ctrl-R) 18 DC3 (ctrl-S) 19 DC4 (ctrl-T) 20 NAK (ctrl-U) 21 SYN (ctrl-V) 22 ETB (ctrl-W) 23 CAN (ctrl-X) 24 EM (ctrl-Y) 25 SUB (ctrl-Z) 26 ESC 27 FS (ctrl-\) 28 GS (ctrl-]) 29 RS (ctrl-^) 30 US (ctrl-_) 31	SPACE 32 ! 33 " 34 # 35 $ 36 % 37 & 38 ' 39 (40) 41 * 42 + 43 , 44 - 45 . 46 / 47 0 48 1 49 2 50 3 51 4 52 5 53 6 54 7 55 8 56 9 57 : 58 ; 59 < 60 = 61 > 62 ? 63	@ 64 A 65 B 66 C 67 D 68 E 69 F 70 G 71 H 72 I 73 J 74 K 75 L 76 M 77 N 78 O 79 P 80 Q 81 R 82 S 83 T 84 U 85 V 86 W 87 X 88 Y 89 Z 90 [91 \ 92] 93 ^ 94 _ 95	` 96 a 97 b 98 c 99 d 100 e 101 f 102 g 103 h 104 i 105 j 106 k 107 l 108 m 109 n 110 o 111 p 112 q 113 r 114 s 115 t 116 u 117 v 118 w 119 x 120 y 121 z 122 { 123 | 124 } 125 ~ 126 DEL 127

Escape Sequence Mode

Many keys on the keyboard you wish to emulate or "type" do not have a "single" character label for a name. Some examples of this might be the "enter," "page up," "backspace," and "shift" keys. You cannot type or emulate these keys in Character Mode, since there is no single ASCII character you can type or transmit from the AAC device to represent them. To type these keys, you must use "Escape Sequence Mode." Escape Sequence Mode simply means to start a transmission of multiple characters with the "escape" character (e.g., ASCII or decimal 27) and end the transmission with a period character (e.g., ASCII or decimal 46). Therefore an escape sequence consists of an ESC character (ASCII 27), followed by a sequence of other ASCII characters (e.g., a Key Name, or GIDEI command), and ending with the PERIOD character (ASCII 46).

For purposes of this document, an "escape" character (ASCII or decimal 27) will be represented by "<esc>." Therefore an escape sequence would look like:

<esc> Key Name .

Key Names are nothing more than a string of ASCII Characters assigned to keys which do not have a "single" ASCII character to represent them. A base set of Key Names which GIDEI2 understands are listed in the Key Name Aliases at the end of this document. (Note, some hardware interfaces may support additional Key Names, which are not part of the GIDEI2 Proposal, but may be incorporated in future standards.)

To make Key Names easier to remember, many Key Names consist of the same characters as the label used for that key on the keyboard. For example, the Key Name for the "backspace" key is either "backspace" or "bspace." The shorter Key Name allows the user to conserve memory when programming that key selection on their alternate input devices, especially if they have a device with several possible selections. Many of the longer Key Names also have shorter equivalents for this purpose. However, longer Key Names are usually available for users who find it easier to learn and remember Key Names in this fashion.

The Escape Sequence Mode has several purposes besides just sending Key Names. Listed below are brief descriptions and examples of Escape Sequence Mode uses. For more explanation on Escape Sequence Mode commands, refer to the next sections.

To Type Special Keys

Keys such as the [Right Arrow] key that have no corresponding ASCII character equivalent are typed using the Escape Sequence Mode. To send the Right Arrow key using Escape Sequence Mode, you must first determine (e.g., look in the Key Names Aliases section) that the Key Name for the Right Arrow key is simply "right." Then you would construct the following Escape Sequence to emulate a press and release of the Right Arrow key.

<esc> right .

To Type Key Combinations

A key combination is where two or more keys are pressed down together (e.g all the keys are pressed before any of the keys are released). The Escape Sequence Mode is used to allow an alternate input device to type key combinations of from two to five keys. For

example, suppose you run a program that requires you to press the shift, control, and F1 keys to perform a particular operation quite frequently. One option might be to use the Escape Sequence Mode "combine" command to perform this multiple key operation. The "combine" command instructs the emulating interface to press the keys in the order in which they are received, and then release them in reverse order. Therefore, the following example instructs the emulating interface to press the shift, control, and F1 keys, and then to release the F1, control, and shift keys. Note, whenever a GIDEI command is used (e.g., "combine" in this example), the comma "," character must be used immediately after the "escape" character and again immediately after the GIDEI command. Refer to the sections later in this document on keyboard escape sequences. mouse escape sequences, and miscellaneous escape sequences for more examples of GIDEI commands used in an escape sequence.

<esc> , combine , shift , control , f1 .

To Type Any ASCII Character

Another purpose of the Escape Sequence Mode is to allow alternate input devices that are unable to produce some ASCII characters, another method to type those that can be generated on the Normal Keyboard of the connected computer system. For example, some "lower-end" or inexpensive alternate input devices may have the ability to transmit characters only, not numbers. Using Escape Sequence Mode, a user with this device could type the number "9" using the following escape sequence.

<esc> nine .

To Operate the Mouse

Besides the keyboard uses described above, the Escape Sequence Mode allows complete control of standard mouse activity, such as moving relative and absolute distances, clicking and double clicking, activating mouse buttons, and dragging. For example, to move the mouse cursor to the right and down on some computer systems, you might create the following escape sequence.

<esc> , move , +25 , +25 .

To Change the Baud Rate

Using the Escape Sequence Mode the user can change the baud rate from the default value of 300 baud. This is an example of one of the miscellaneous GIDEI commands that are available. The baud or "baudrate" has to do with how fast information is communicated across the serial link. The higher the "baudrate," typically the faster the response.

<esc> , baudrate , 1200 .

Clearing Syntax Errors

The Escape Sequence Mode is used to clear errors resulting from improper use of syntax.

<esc> .

Characters Allowed in GIDEI Escape Sequences

While GIDEI2 supports the transmission of Extended ASCII characters 128 through ASCII 255, the make-up of character strings used in GIDEI Escape Sequences remain limited to the printable ASCII alphanumeric characters (ASCII 32 - 126 decimal, see Table 1). Therefore, if a user wishes to transmit an Extended ASCII character in the range from ASCII 128 through ASCII 255, either their alternative input device must be capable of generating and sending that particular ASCII character (e.g Character Mode), or they must use a GIDEI Escape Sequence which contains the printable ASCII Key Name for that particular ASCII character (e.g., see GIDEI Key Name Aliases section at the end of this document).

For example, if the user wished to get the "A with circumflex" or "â" character, which is ASCII character 131 on some DOS systems, their alternate input device must be capable of transmitting Extended ASCII or decimal "131." If their device is unable to generate ASCII characters above 127, they must then use an escape sequence after determining the proper Key Name for the "A with circumflex" character in the Key Name Aliases table (see GIDEI Key Name Aliases at the end of this document).

Note the following examples.

If the AAC device could transmit Extended ASCII in GIDEI2.

ASCII character 171 = [½] , ASCII character 131 = [â]

--or--

If the AAC device is unable to generate Extended ASCII characters, it could use the Key Name for the "onehalf" character in an escape sequence as follows: <esc> onehalf. Similarly, the escape sequence for â is <esc> acircumflex.

The following is an example of a character NOT allowed in GIDEI Escape Sequence Mode:

<esc> ce'nt .

This is NOT allowed in Escape Sequence Mode. The "e" with the "acute" accent is not a printable ASCII character within the range of ASCII 32 through ASCII 126 (see Table 1) and therefore is not allowed inside an escape sequence (e.g., between the ESC and PERIOD characters).

Please note that while the GIDEI2 Proposal supports transmission of the Extended ASCII characters between ASCII 128 through ASCII 255 in Character Mode, it is still up to the hardware interface (either the "black box" connected to the computer or the emulation interface software) to create the proper character or keystroke at the receiving computer.

Capitalization

All characters in escape sequences are converted to lower case by most hardware or software interfaces prior to processing.

Spaces

Space characters (decimal 32) anywhere in the escape sequences are ignored by most hardware or software interfaces.

Format

Escape sequences use one of two possible syntaxes, which may not be evident to most users but is described here for GIDEI Proposal completeness. Escape sequences can either include the appropriate GIDEI Command name in the escape sequence or they can "imply" a particular command is being used. The following two examples are both escape sequences involved with the keyboard, however only one requires the use of a GIDEI command name

Including a GIDEI Command Name

Only one GIDEI Command Name can be included in an escape sequence to distinguish what action the hardware interface is to perform. Parameters are also included where appropriate and are separated with a comma character (ASCII or decimal "44").

<esc> , hold , shift . a

Note the hold GIDEI command name, preceded by a comma.

Also note that two keys are "typed," the "shift" key and the "a" key.

Without a GIDEI Command Name, or the "implied" method

The Implied method is typically used to type single keys. The command name is NOT specified in the escape sequence (e.g., it is implied).

<esc> return .

Note you DO NOT need the GIDEI "hold" command or comma "," separator (ASCII or decimal "44") to press a single key (e.g., Key Name).

Error Handling

Determination of Invalid Fields

The hardware interface tests each field in a GIDEI Escape Sequence to determine if it has invalid character strings.

Field Defined

A field consists of the characters in an escape sequence between the delimiting characters escape, comma, and period.

Invalid Fields

Invalid fields are fields with characters that do not conform to any of the formats (e.g., GIDEI command names or Key Name aliases) defined for escape sequences. Please

review the previous section titled "Characters Allowed in GIDEI Escape Sequences." When the emulating interface determines that a field contains an unrecognized character string, it may provide some form of user feedback (e.g., audible sound) and it then switches to Character Mode and types the remaining characters in the invalid field followed by any other characters received.

Invalid Commands or Unsupported Features

Invalid commands or parts of commands which are unsupported via GIDEI should provide the user with some form of feedback (e.g. audible and visual, if possible), and then the emulating interface should switch to Character Mode and type the remaining characters in the invalid field followed by any other characters received if possible. It is possible that some kind of reset may also be necessary, such as an escape sequence reset as described in the next section, or possibly needing to re-establish serial communications (e.g. see prior section on Hardware Connection).

Resetting the Escape Sequence Mode

If the hardware interface is in Escape Sequence Mode and receives another escape character before getting the period character (e.g., ASCII or decimal "46") which terminates an escape sequence, then all the characters before this escape character are discarded. After discarding previous characters, the hardware interface remains in Escape Sequence Mode ready to start assembling a new escape sequence.

Command and Parameter Designations

Aliases

Each GIDEI command, Key Name, or mouse action is assigned a character or string of characters which specify it. This basically gives each GIDEI command, Key Name, or mouse action a "name" used to refer or identify it. Sometimes this is also called an "alias," since many of the keys on the keyboard have more than one "name." For example, the "return" key is also called the "enter" key on some keyboards. In the list of GIDEI Key Name Aliases at the end of this document, the "return" Key Name also has the names or "aliases" of "ret" and "enter."

Delimiting Characters within Escape Sequences

While operating a computer with an alternative input device, it may be necessary in an escape sequence to type the escape, space, period, or comma key on the computer. Since these four keys have a special meaning (space is ignored) in the GIDEI Escape Sequence Mode, it would seem difficult to type them. However, the following examples show you how to use these four characters within an escape sequence to generate their respective keys.

The Escape Key

The [Esc] key on the computer system must be designated by the characters of the Key Name, esc or escape.

<esc> esc. --or-- <esc> escape .

The Period Key

The [.] key on the computer system must be designated by the Key Name, period.

<esc> period .

The Space Key

The [space] key on the computer is designated by the Key Name, space.

<esc> space .

The Comma Key

The [comma] key on the computer is designated by the Key Name, comma.

<esc> comma .

Miscellaneous Escape Sequence Commands

baudrate

The baudrate command allows the user to change the baud rate of the serial port. This rate affects both the transmission of characters to the hardware interface and the transmission of XON and XOFF characters from the hardware interface to the alternate input or AAC device. The possible baud rates are 300, 1200, 2400, 4800, 9600, and 19200.

Format

<esc> , baudrate , XXXXX .
where XXXXX is 300, 1200, 2400, 4800, 9600, or 19200.

Additional Information

When the hardware interface is ready to receive at the new baud rate, it sends an XON character at the new baud rate and sets the CTS line on the serial port. The alternate input device must wait for these signals before transmitting at the new baud rate. Please refer to the section on "Hardware Connection" for more information regarding other parameters which are part of the baudrate or transmission speed of the characters between the alternate input or external AAC device and the hardware interface.

Keyboard Escape Sequence Commands

combine

The combine command allows the user to type key combinations of up to five keys using a single escape sequence. Key combination means two to five keys pressed in the order in which they are presented in the escape sequence with all keys eventually held down simultaneously. They are then released in reverse order.

Format

<esc> , combine , Key Name 1, Key Name 2, Key Name 3, Key Name 4, Key Name 5.

Example

<esc> , combine , control , alt , delete .
Note, this is a common three key combination used to "warm" boot or restart personal computers.

Additional Information

Only Key Names are allowed in this escape sequence. Please see the Key Name Aliases table for a list of the valid key names for a given keyboard and computer.

While this command is supposed to be used for a minimum of two keys, it is not an error if only one Key Name is presented in the escape sequence.

Some key combinations may be impossible to type simultaneously due to the nature of a given keyboard or computer. The hardware interface will only be required to simulate keystrokes that are possible on the normal keyboard.

hold

The hold command allows the user to "hold" from one to five keys down together with the next key the hardware interface is instructed to type. This is used primarily for modifier keys like the "control" key and the "shift" key which are used in combination with another key.

The function of this command differs from the combine command in that it allows the user to program generic hold keys on the alternate input device rather than programming all the possible combinations with the combine command. Once the generic key is programmed, it would only take two selections to type any two key control or shift combination.

Format

<esc> , hold , Key Name 1, Key Name 2, Key Name , Key Name 4, Key Name 5.

Examples

<esc> , hold , shift . a
Instructs the hardware interface to press the shift key, then press an "a" key, which should produce an upper case "A", and then release the "a" key, and then finally release the "shift" key.

<esc> , hold , control .

Instruct the hardware interface to press and hold the "control" key until the next single character (e.g., Character Mode) is transmitted. In this

manner, the user of an alternative input device can program a selection on their devices for common keys like the "shift" or "control" and combine these keys to produce the necessary output from several other keys using any two selections. For example:

<esc> , hold , control . c

will produce a "^C" on some disk operating systems.

Additional Information

Keys are typed by the hardware interface as follows: Each key (referred to by their Key Name) that is received after the hold escape sequence command is pressed and held down in that order. After the last Key Name is received and pressed, then the Key Names are released in reverse order.

Only Key Names are allowed after the hold command in this escape sequence. Please see the section on Key Name Aliases for a list of the valid key names for a given keyboard and computer.

Some key combinations may be impossible to type simultaneously due to the nature of a given keyboard or computer. The hardware interface will only be required to simulate keystrokes that are possible on the normal keyboard.

If the Hold sequence is sent by mistake, you may clear it with the rel command. (See rel later in this section.)

Example:

If I wanted to type a capital g key I could send or program into one selection:

<esc> , hold , shift. g

--or--

The <esc> , hold , shift. sequence would be programmed into one selection on the alternate input device so it would only take one selection for the shift and a second for the g, or any other letter you wished to "capitalize."

If I wanted to type a control c , I could send or program into one selection:

<esc> , hold , ctrl. c

--or--

Again, once the control escape sequence is programmed into one selection, any other control key combination would take only two selections on the AAC device.

lock

The lock command allows the user to press a key or keys (locking key[s] down) with out releasing key(s).

Format

<esc> , lock , Key Name 1, Key Name 2, Key Name 3, Key Name 4, Key Name 5.

Example

<esc> , lock , shift .

This escape sequence instructs the hardware interface to press and hold pressed the "shift" key until a "rel" command is transmitted. This action should cause all subsequent transmissions of alphabetic characters to be upper case.

<esc> , lock , kp8 .

Some hardware interfaces, such as the SerialKeys feature of AccessDOS are capable of supporting mouse actions from the keyboard, through a feature called "MouseKeys." Using MouseKeys, a keyboard user can perform all mouse functions from the numeric keypad. For example, pressing the key pad key "5" performs a mouse click while pressing the key pad "8" key causes the mouse cursor to travel upwards. An alternative input or AAC device user can also make use of the MouseKey feature, if the hardware interface supports them. In the case of AccessDOS, locking the key pad key "8" would also cause the mouse cursor to move upwards, until the "rel" command was transmitted, provided MouseKeys was turned On. This is one possible alternative method to provide continuous mouse cursor motion.

Additional Information

This command, in combination with the rel command, may be used to send explicitly defined key downs and key ups for any given key.

If the lock sequence is sent by mistake, you may clear it with the rel command. (See rel command)

Only Key Names are allowed in this escape sequence. Please see the section on Key Name Aliases for a list of the valid Key Names for a given keyboard and computer.

Some key combinations may be impossible to type simultaneously due to the nature of a given keyboard or computer. The hardware interface will only be required to simulate keystrokes that are possible on the normal keyboard.

rel

The rel command clears all previous "Lock"ed keys or any subset of them and all "Hold"ed" keys .

Format

<esc> , rel .
Release all "locked" or "hold"ed" keys.

<esc> , rel , Key Name 1, Key Name 2, Key Name 3, Key Name 4, Key Name 5.

Instructs the hardware interface to release the specific Key Names only. Note, all Hold"ed are also released.

Additional Information

Implied Press

The Implied Press sequence is an escape sequence without any explicit GIDEI command name being given. It causes the key designated in the sequence to be typed. It is used to type keys or functions which have no ASCII equivalent and therefore cannot be sent in Character Mode. Many of the key names in the Key Name Alias table can be used in this fashion.

Format

<esc> Key Name .

Example

<esc> esc .

The implied press used to type the "escape" key is shown.

<esc> pageup .

The implied press is used to type the "page up" key is shown.

Additional Information

Mouse Escape Sequence Commands

moureset

The moureset command will cause the mouse cursor to release any locked buttons, move its position to the upper left hand corner of the screen, and reset its internal X-Y position counter to (0,0).

Format

<esc>, moureset .

Additional Information

This command should be given before any other mouse command to insure the mouse parameters are initialized and that internal mouse coordinates are set to (0,0).

anchor

The mouse anchor command should be used to identify locations on the screen that the user wishes to return the mouse cursor to on a frequent basis. An example of this might be a drawing program, where the user may wish to return to a drawing tool palette often, and therefore setting an "anchor" at the palette location would facilitate easy return to the palette. For example, a user could set an anchor in their work, exit their work with a single goto command and return to a preset anchor position with in the drawing palette, select a new tool, and then use another single goto command to return to the exact spot they left their work.

Formats

<esc> , anchor .

Sets or resets the current mouse cursor screen location to be the anchor position the user is about to name. Any lowercase letter "a" through "z" can be used to "name" the current screen position of the mouse cursor. This provides 26 separate "anchor" locations. The next "single" lowercase letter entered after the "anchor" command becomes the "name" of the current mouse cursor screen position. This also means programmable communication devices need to program just one selection for the "anchor" command, since many of these devices already have the lowercase letters "a" through "z" pre-programmed.

Examples

<esc> , anchor . h .

Instructs the emulation interface to set a mouse anchor point at the current coordinates of the mouse cursor on the screen and gives that location "h" as a label or "name." Now if the user were to move the mouse cursor around, and then wishes to return to this location, they would use the following command:

<esc> , goto . h

Instructs the emulation interface to return the mouse cursor to preassigned location or name "h."

Additional Information

mougo

The mouse mougo command is used to move the mouse cursor in a specified direction, at a specified speed, in a continuous motion until stopped. Note, not all hardware interfaces will support the "mougo" command, especially if they also support some other means of providing a continuous motion mouse cursor.

Format

<esc> , mougo , YY, XX .
where YY is a direction and XX is a speed.

Allowable mouse cursor directions "up", "down", "left", and "right" which define the horizontal and vertical directions, while "upleft", "upright", "downleft", and "downright" would define the four diagonal directions.

Allowable speeds are a given with a number from 1 through 10, with faster mouse cursor speeds being associated with larger numbers (e.g., 1 is the slowest speed, and 10 would be the fastest speed).

Example

<esc> , mougo, downleft, 5 .
Instructs the emulation interface to move the mouse cursor in a continuous diagonal motion down and to the left from its current location at a speed of 5, which is half speed.

Additional Information

moustop

The mouse moustop command is used to stop the mouse cursor when is was started in continuous motion by the mougo command. There are no parameters to the moustop command. (See note under "mougo" command, regarding mougo command support. If the mougo command is not support by the hardware interface, then the moustop command would likewise not need to be supported.)

Format

<esc> , moustop .

Additional Information

moulock

The mouse moulock command allows the user to press down and hold (e.g., lock) a default or specified mouse button identifier(s).

Many operating systems define the left-most mouse button as mouse button identifier #1, and the remaining mouse buttons are identified from left-to-right in ascending order as the default. However, these same operating systems also allow the user to reassign mouse button actions, such that the "left" or mouse identifier #1 may not physically be located on the left-most side of the mouse puck or track ball. Therefore, mouse button identifier now refers to a mouse button action, such as pressing mouse button #1, rather than a location, such as pressing the "left" mouse button.

Therefore, users are encouraged to follow the new button identifiers, rather than previous GIDEI conventions which defined a "left" or "right" mouse button.

Format

<esc> , moulock .
to lock the default (e.g., currently active or assigned) button, or

<esc> , moulock , ???? .

<esc> , moulock , X .

<esc> , moulock , X , X , X , X , X .

Up to five buttons can be locked down using the "moulock" command.
where ???? is:
either "left" or "right" to refer to the left button or the right button. Note, the "left" mouse button may not necessarily be the button located on the "left" side of a mouse puck, since many operating systems now allow the user to reassign mouse button actions. Therefore, users are encourage to switch to the button number method of button identification as follows.

where X is:

button numbers "but1" through "but5", as mouse button identifier(s), whose action is defined by the operating system.

Additional Information

mourel

The mouse mourel command releases all or specified mouse buttons after they have been locked.

Format

<esc> , mourel .
to release the all buttons, or

<esc> , mourel , ???? .

<esc> , mourel , X .

<esc> , mourel , X , X , X , X , X .

Up to five buttons can be released using the "mourel" command.
where ???? is:
either "left" or "right" to refer to the left button or the right button.

where X is ;

button numbers "but1" through "but5", as mouse button identifier(s), whose action is defined by the operating system.

Additional Information

See notes under mouse mourel command for more information regarding "left" mouse button and mouse button identifier #1.

click

The mouse click command will cause the specified button to be locked down and then immediately released. This action is known as a click.

Format

<esc> , click .
to click the default button, or

<esc> , click , ???? .

<esc> , click , X .

where ???? is:
either "left" or "right" to refer to the left button or the right button.

where X is ;

button numbers "but1" through "but5", as mouse button identifier(s), whose action is defined by the operating system.

Additional Information

See notes under mouse mourel command for more information regarding "left" mouse button and mouse button identifier #1. Also note, some systems may allow you to click more than one mouse button at a time.

dblclick

The mouse dblclick command will cause the specified button to be double clicked. That is, the button is locked down and then immediately released, locked down once again, and immediately released.

Format

<esc> , dblclick .
to double click the default button, or

<esc> , dblclick , ???? .

<esc> , dblclick , X .

where ???? is:
either "left" or "right" to refer to the left button or the right button.

where X is:

button numbers "but1" through "but5", as mouse button identifier(s), whose action is defined by the operating system.

Additional Information

See notes under mouse mourel command for more information regarding "left" mouse button and mouse button identifier #1. Also note, some systems may allow you to double click more than one mouse button at a time.

goto

A. The mouse goto command will cause the mouse cursor to move to the specified absolute screen position. The position is specified as X and Y coordinates and represent single pixels on the computer screen. Both X and Y values must be greater than or equal to 0. On most computers, the origin of the axis is the upper left hand corner of the screen. Therefore, on most systems, X values increase as you move to the right, and the Y values increase as you move down.

The X and Y values given with the mouse goto command may also be written with a "+" sign as part of the number.

Format

<esc> , goto , X coordinate , Y coordinate .

Example

<esc> , goto , 25 , 25 .

--or--

<esc> , goto , +25 , +25 .

Instructs the emulating interface to direct the mouse cursor to go to location X equals 25 and location Y equals 25 on the current display.

B. The mouse goto command will also cause the mouse cursor to move to the specified anchor location when the location coordinate is specified as the next "single" lowercase letter "a" through "z."

Format

<esc> , goto .

Example

<esc> , goto . a

Instructs the emulating interface to return the mouse cursor to "anchor" location "a" assuming it was predefined using the anchor command (see anchor command). Lowercase letters "a" through "z" are valid anchor "names" or labels, allowing for 26 separate anchor location points. This also means programmable communication devices need to program just one selection for the "goto" command when used to return to "anchor" points, since many of these devices already have the lowercase letters "a" through "z" pre- programmed.

Additional Information

move

The mouse move command will cause the mouse to move the specified number of units in the X and Y directions. Positive values "+" will move the mouse cursor to the right or down, negative values "-" will move the mouse cursor to the left or up. Since the mouse move command can move the mouse cursor in both a positive or negative direction, as defined by the receiving computer, the X and Y values given with the mouse move command must include either the "+" or "-" signs with each number.

Format

<esc> , move , units in +/-X direction , units in +/-Y direction .

Example

<esc> , move , +25 , -25 .

Instructs the hardware interface to move the mouse cursor 25 "pixel" points in a positive "X" direction and 25 "pixel" points in a negative "Y" direction. Many computers would understand this to mean to move the mouse cursor to the right and upward from its current location. Your system may vary depending upon how it defines mouse cursor motion.

GIDEI Key Name Aliases

List of Currently Defined Key Names

A	B	C	D	E	F
aacute	backslash	cacute	dblquote	eacute	f1
acircumflex	backspace	cancel	del	ecaron	f10
acute	bbar	capslk	delete	ecircumflex Ê ê	f11
adieresis	break	capslock	dieresis ¨	edieresis Ë ë	f12
ae	bslash	ccaron	divide	egrave	f13
again	bspace	ccedilla	dn	eight	f14
agrave		cedilla	dollar	end	f15
alphanumeric		circumflex	down	enter	f16
alt		clear		equal	f17
altgr		cmd		esc	f18
amp		colon		escape	f19
ampersand		comma		eth	f2
aogonek		command		exclaim	f20
apostrophe		compose		exclaimdown	f21
apple		control		execute	f22
appskey		conversion			f23
aring		copy			f24
ast		ctrl			f3
asterisk		cut			f4
at					f5
					f6
					f7
					f8
					f9
					find
					five
					four
					front
					fullsize

G	H	I	K	M	N
graphical grave	help hiragana home hyphen	iacute icaron icircumflex idieresis igrave ins insert	kana kanji kanjidict kp* kp+ kp- kp/ kp0 kp1 kp2 kp3 kp4 kp5 kp6 kp7 kp8 kp9 kp= kpcomma kpdel kpdivide kpdn kpdown kpdp kpend kpenter kpequal kphome kphyphen kpins kpinsert kpleft kpmidl kpminus kpperiod kppgdn kppgup kpplus kpright kpslash kpstar kptimes kpup	menu meta micro minus mordinal multiply	ncaron next nine noconversio n ntilde number numlk numlock

O	P	R	S	T	U
oacute ocircumflex odieresis oe ogonek ograve ohungaruml aut one onehalf onequarter ooblique open openapple option otilde	pagedown pageup paste pause period pf1 pf2 pf3 pf4 pgdn pgup plus pound prev print printscreen props prtscr	ralt rbrace rbracket rcaron rcmd rcommand rcompose rcontrol rctrl remove reset ret return right rightwinkey ring rolldown rollup romanize ropenapple roption rparen rshift	sacute scaron scroll scrolllock section select semicolon seven sharps shift shiftleft shiftright six slash small space stop superone superthree supertwo sysreq	tab tcaron three tilde triangle two	uacute ucircumflex udieresis ugrave uhungarumlaut Ű underscore undo up uring

W	Y	Z
wordreg wordrem	yacute ydieresis yen	zcaron zdotaccent zero

Appendix A

The Plug and Play Architecture

The Plug and Play initiative is an industry-wide effort that promises to make PC installation and use easier for computer users.

Plug and Play is both a design philosophy and a set of PC architecture specifications. The ultimate goal of Plug and Play is to design intelligence into the PC itself, to handle installation and configuration tasks without user intervention. A Plug and Play system has a number of characteristics. First, any installation is a simple, fail-safe operation. For devices the installation is automatic: plug the device in, turn on the system, and it works. With a Plug and Play system, the user can insert and remove devices, or connect-to or disconnect-from a docking station or network, without restarting the system or fiddling with configuration parameters. The system determines the optimal configuration, and applications automatically adjust to take full advantage of the new configuration. Users do not need to modify expansion card jumper settings, or even modify operating system configuration files. The benefits to both users and computer industry members will be substantial, as PC ease-of-use will be enhanced while support costs will be substantially lowered.

Plug and Play COM

The Plug and Play COM specification, a component of the overall Plug and Play Architecture, presents a mechanism to provide automatic configuration capability to peripheral devices that connect to a PC using Asynchronous Serial Data Interchange on standard serial ports, commonly known as COM ports. This enables full Plug and Play for the PC system, including external serial peripherals, referred to here as COM Devices.

The essential elements of Plug and Play COM are:

Detect attachment of serial devices

Identify the device

Locate a driver for the device

Detect detachment of serial devices

Full compatibility between existing COM hardware and software and the new hardware and software which support Plug and Play

Device Implementation

The full Plug and Play COM specification defines mechanisms that each Plug and Play COM device must implement to support identification, which in turn supports automatic configuration of the entire PC system without end-user intervention.

The solution of the COM device identification problem addresses major concerns of end-users, system integrators, and operating system vendors. It also presents an excellent business opportunity for providing measurable value and differentiation in the short term, by reducing the user difficulties and associated technical support costs.

There are two ways that serial device manufacturers can implement this specification. For simple hardware based devices (e.g., serial mice) the hardware (e.g., ASICs) will need revision. For intelligent serial devices (e.g., modems and printers) these changes can be implemented by controller code firmware changes. However, care has to be taken in the controller code to avoid false-positive (erroneous) detection of the COM Enumerator (confusing the application software with unexpected PnP ID) or failure to detect the COM Enumerator.

References

There is now a forum on the Compuserve Information Service to address Plug and Play in Personal Computers. Please refer to this forum for all your Plug and Play support needs. This forum is the official mechanism for support, information and discussion, as well as receiving current versions of Specifications. Type "GO PLUGPLAY" to access this forum. To obtain a Compuserve account (if you don't already have one) please call 1- 800-848-8199.

For further details on the Plug and Play architecture and implementation details, see the following documents, some may be available on the CompuServe Plug and Play forum:

Plug and Play External COM Device Specification, Microsoft 1994

Plug and Play ISA Specification, Microsoft 1993

Plug and Play BIOS Specification, Microsoft 1993

Plug and Play PCMCIA Specification, Microsoft 1993

CCITT V.24, List of definitions for interchange circuits between data terminal equipment and data circuit terminating equipment.

EIA/TIA-232-E Interface between data terminal equipment and data communication equipment employing serial binary data interchange on unbalanced circuits.

EIA/TIA-574 9-position Non-synchronous Interface Between Data Terminal Equipment and Data Circuit- terminating Equipment Employing Serial Binary Data Interchange

EIA/TIA-578 Asynchronous Facsimile DCE Control Standard-Service Class 1

TIA/EIA-592 Asynchronous Facsimile DCE Control Standard-Service Class 2

TIA/EIA-602 Serial Asynchronous Automatic Dialing and Control

PCCA XSTD-101 Data Transmission Systems and Equipment- Network Independent Dialing, Control and Data Transfer for Asynchronous DCE

Version Changes

Version 2.1

-Corrected HOLD command syntax in examples showing holding shift-key with A key.

-Added new key names "appskey", "leftwinkey", "rightwinkey" for compatibility with AAC Keys

Version 2.2

-The following title page contact information is replaced:

Standards Project Manager
Trace Research and Development Center, University of Wisconsin
2107 Engineering Centers Bldg., 1550 Engineering Dr., Madison, WI 53719

-The document has been repaginated, page numbers added, and minor spelling changes made that do not alter the meaning of specified item.

- In **Characters Allowed in GIDEI Escape Sequences**, Second paragraph the following substitutions in order of occurrence: " â" *was* "1" ; 131 *was* 177 (two places).
ASCII character 171 = [½] , ASCII character 131 = [â] *was* [one-half-symbol]
<esc> onehalf -- *was* --<esc> acircumflex. Added: Similarly, the escape sequence for â is <esc> acircumflex.

-In moureset command format: <esc>,moureset. *was* ,moureset.

The link for Trace R&D Center Web site *was* http://trace.wisc.edu/

trace.wisc.edu which followed below, is replaced.

Trace RERC

This document is hosted on the Trace R&D Center Web site. Please visit our home page for the latest information about Designing a More Usable World - for All.