Universal Logging Format for AAC

Preliminary Logfile Specification
Last Updated: January 4, 2004

This page describes a logfile format proposed by Enkidu Research, Inc. and Dr. Jeff Higginbotham of State University of New York at Buffalo's Communication and Assistive Device Laboratory. The development of this format has been funded in part by the National Institute on Disability and Rehabilitation Research's (NIDRR's) Rehabilitation's Engineering Research Center on Communication Enhancement (the AAC-RERC, grant #H133E980026). However, the opinions expressed are those of the authors and do not necessarily reflect those of the supporting agency.

Readers of this document are expected to have familiarized themselves with the basics of the proposed logfile format through the CSUN 2000 paper: A Universal Logging Format for Augmentative Communication. Please direct all comments and suggestions regarding the proposed format to logfile@enkidu.net.

GENERAL FORMAT

The basic structure of the proposed standard is straightforward. An optional header defines the type of information that will be provided in the logfile. The header (if present) is followed by an arbitrary number of logfile entries which corresponds to distinct events in the language production process. Each entry is subdivided into different fields (typically separated by spaces) that provide details on different aspects of the event. For example, the TIME field indicates the time at which the event occurred and the OUTPUT field describes the output (if any) associated with the event. The order and type of each field may be specified in the header, or may be determined dynamically based on the field specifiers that appear in the entry. For example, if we were logging TIME and OUTPUT our logfile might look like this (lines beginning with # are comments):

```
#Start logfile
#Header information
TIME
OUTPUT
#Logfile entries
11:20:14 "H"
11:20:22 "e"
11:20:40 "l"
11:20:50 "Hello"
```

Or, if we prefer we could omit the header and use specifiers (or their default abbreviations, T=TIME and O=OUTPUT):

```
#Start logfile
#Logfile entries
TIME:11:20:14 OUTPUT:"H"
TIME:11:20:22 OUTPUT:"e"
T:11:20:40 O:"l"
T:11:20:50 O:"Hello"
```

In addition, we could choose to use a default style that required neither headers nor field specifiers. For example, the LAM format (TIME and OUTPUT) shown above is actually one of the defaults, so stripping the header from the first example would yield a valid logfile format. In cases where storage is not an issue, we recommend using both headers and field specifiers to make the logfiles as readable as possible.

As mentioned above, each entry corresponds to a "distinct event in the language production process". We have deliberately been vague in this definition. Exactly what constitutes such an event is largely dependent upon the interface, the type of information being collected, and a range of other factors. The proposed format does not mandate what types of events should be recorded, it merely specifies the format of those events which are added to the logfile.

Consider an example in which a single-switch scanning interface is used to access a Minspeak-like interface. The manufacturer may choose only to log events that happen to result in an output - the selection of the final symbol in a Minspeak sequence. The LAM takes this approach out of necessity, since this is the only information generally available on the serial port of AAC devices. Alternatively, the manufacturer might decide to log output events *and* individual symbol selections. In this case, the logfile provides a more detailed analysis of how the user arrived at the output. We would know, for example, not only that the user said the word "purple", but also how he or she generated the word. We could answer questions such as "Did the user make any mistakes?" and "Did the user use the most efficient sequence?". At an even finer grain of logging, the manufacturer might decide to record individual scanning events (for example, "scan advance" and "scan select") along with the selection and output information. Now we could answer questions such as "How many times did the user select an incorrect scanning group?" and "How often did the user miss the correct scanning group?".

In many interfaces, a single user selection can result in multiple events of interest. In a scanning interface, for example, we might want to log the fact that the user hit the scanning selection switch as well as logging the result of that switch. As another example, when the user selects the final symbol in a Minspeak sequence, we'd like to be able to record the symbol that was selected, but we'd also like to log the fact that the system expanded a multi-symbol sequence into an output word. Rather than try to squeeze too much information in a single entry, these events can be split into multiple entries. For the Minspeak example, we might see logfile entries like below:

```
T:12:29:34 O:"" B:"color" A:KYD  # user selects "color" button
T:12:30:12 O:"" B:"apple" A:KYD  # user selects "apple" button
T:12:30:12 O:"red " B:"" A:ASE   # system expands sequence to "red"
```

The ACTION (A) field indicates the type of action associated with an event. In this case,the user made two KEYDOWN (KYD) actions to select the "color" and "apple" symbols. The system then expanded the two-symbol sequence to the word "red", as indicated by the AUTO_SEQUENCE_EXPAND (ASE) action (and the time field identical to that of the previous entry). For devices and software that support sophisticated macro coding (for example, Speaking Dynamically), there may be many entries that correspond to the same fundamental event.

The proposed format may seem rather complex at first glance. It is important to note that we *are not* proposing that manufacturers support all of the variants and features, but rather that they adopt a style that is compatible with the proposed standard. We expect that most manufacturers would choose a fixed format that utilizes some fixed subset of the standard that would be appropriate for their devices. Only programs designed to process and analyze logfiles (such as our ACQUA software) would need to support the logging format in its entirety.

ENTRY FIELDS

There are 12 standard logging fields in the proposed standard. Each of these fields is assigned a unique specifier (such as TIME) and a default abbreviation (T for TIME). The abbreviation may be redefined to an arbitrary string of 1 or more characters in the header (see below). Additionally, new fields (and abbreviations for these fields) may be defined in the header and used in the logfile entries.

The type of data associated with a field varies. For some fields, such as TIME, a numeric value is assigned. For others, such as OUTPUT, an arbitrary character string may appear. For some fields, such as ACTION, the field data may take one of a limited set of descriptors (such as KYD and ASE in the example above). For fields that take descriptors, a set of default descriptors and their abbreviations are defined. Additionally, new descriptors and alternative abbreviations may be defined in the header (see below).

Within each entry, fields are separated by one or more spaces or tabs. If a field specifier is used, then it must be followed immediately by a color and then by the field data (with no intervening spaces). If field data contains any spaces or tabs, then the data should be enclosed in quotes (as indicated in the examples above) to delimit the extent of the data. Even if there are not spaces or tabs, the quote delimiters may be used for consistency. As in the header, lines that begin with a # are ignored.

TIME (T)

Perhaps the most fundamental of all fields, the timestamp indicates the time at which an event occurred. Arbitrary time resolutions, from years down to milliseconds, are supported through the following generic format:

YYYY:MM:DD:HH:MM:SS.XYZ

Time subfields may be removed from either end of this construct. Thus DD:HH:MM (days, hours, and minutes) and SS.X (seconds and tenths of seconds) both represent valid times. When any subfields are removed from the right side (with the exception of the fractional seconds), the time format must be specified in the header. The range of each subfield is limited to the standard range for that field (for example, minutes can range from 0 to 59 and hours can range from 0 to 23), with the exception that the leftmost subfield has no upper limit. In the DD:HH:MM format, for example, 45:08:12 would be an acceptable timestamp (indicating 45 days, 8 hours, and 12 minutes elapsed since the start of the logfile). Note that if months are the leftmost field, some ambiguity may arise from the fact that the year remains unknown (i.e., leap years can throw off subsequent analysis).

By default, the time field is given as an absolute value meant to represent the actual time (ABSOLUTE mode). However, this field can also be given as a relative value with respect to either the beginning of the logfile session (RELATIVE mode) or with respect to the previous logfile entry (DELTA mode). The relative modes are particularly useful for hardware devices that do not have access to absolute time. For non-absolute times, the leftmost field must not be higher than days since using months (or years) would result in ambiguities (i.e., not every month has the same number of days).

OUTPUT (O)

The OUTPUT field indicates the text output (if any) associated with a particular logfile entry. The text should generally be delimited by quotation marks (since it may contain spaces). Control characters, such as backspace, are indicated by backslashes (\) using standard conventions (for example, \b for backspace and \n for newline). Quotations marks are indicated by \" and the backslash itself is indicated by \\. For compliance with Romich and Hill's LAM format, non-textual sequences may be encoded in the output field using *[and]* delimiters. In the LAM, control sequences are generally indicated in this manner. Note that other fields (such as the TYPE field) are used for this purpose within the proposed standard.

ACTION (A)

The ACTION field describes the type of event that precipitated the logfile entry. The event may be related to an explicit user action, such as a keypress, or to an implicit device action, such as the advancement of a scanning group during autoscanning. In either case, the action may or may not be associated with an output. For example, if a user hits a key on a traditional keyboard then the keypress action will have a corresponding output. However, if the user hits a key on a Minspeak keyboard, the action will have no output unless the key corresponds to the final symbol in the Minspeak sequence. Actions are not meant to indicate the purpose of a logfile entry, merely the kind of user or device action that precipitated the entry. For example, an action of type SWITCH1_DOWN would indicate that the user activated switch 1, but would not provide information about the consequences of this action. For this information, we would have to look at the TYPE field (for example, to find out that the switch corresponds to a scanning selection).

The ACTION field is designed to take a predefined descriptor, although additional descriptors can be defined in the logfile header (see below). Actions may be specified with the entire descriptor or with a predefined three-letter abbreviation. The action abbreviations can also be changed in the logfile header.

LMD	LEFT_MOUSE_DOWN	Left mouse button clicked down
MMD	MIDDLE_MOUSE_DOWN	Middle mouse button clicked down
RMD	RIGHT_MOUSE_DOWN	Right mouse button clicked down
LMU	LEFT_MOUSE_UP	Left mouse button released
MMU	MIDDLE_MOUSE_UP	Middle mouse button released
RMU	RIGHT_MOUSE_UP	Right mouse button released
LM2	LEFT_MOUSE_DOUBLECLICK	Left mouse double-clicked
MM2	MIDDLE_MOUSE_DOUBLECLICK	Middle mouse double-clicked
RM2	RIGHT_MOUSE_DOUBLECLICK	Right mouse double-clicked
MMV	MOUSE_MOVE	Mouse moved
KYD	KEY_DOWN	Keyboard key pressed down
KYU	KEY_UP	Keyboard key released
KYR	KEY_REPEAT	Keyboard key autorepeat
TSD	TOUCHSCREEN_DOWN	Touchscreen key pressed down
TSU	TOUCHSCREEN_UP	Touchscreen key released
TSR	TOUCHSCREEN_REPEAT	Touchscreen key autorepeat
		Switch 1 pressed down (S1D

S1D	SWITCH1_DOWN	Switch 1 pressed down (S1D through S5D)
S1U	SWITCH1_UP	Switch 1 released (S1U through S5U)
JSM	JOYSTICK_MOVE	Joystick moved
TBM	TRACKBALL_MOVE	Trackball moved
ADW	AUTO_DWELL	Automatic dwell timer
ATM	AUTO_TIMER	Autoscanning timer
ADS	AUTO_DISAMBIGUATE	Automatic disambiguation
ACP	AUTO_CAPITALIZE	Automatic capitalization
ASP	AUTO_SPACE	Automatic space insertion
ASE	AUTO_SEQUENCE_EXPANSION	Automatic expansion of predefined sequence
AUN	AUTO_UNKNOWN	Automatic entry, origin unknown

Descriptors that begin with AUTO are not associated with an explicit user action, but nevertheless are important parts of the language production process. For example, it may be important to log timer events in a single-switch scanning interface.

The ACTION field can be supplemented by additional information about the action. This is done by concatenating the information after the action identifier, using a period (.) as a separator. For example, we might want to indicate the screen position of a left mouse click. This might look like: LMD.56.128. There is no standard for such supplemental information, but its existence should not influence any logfile processing.

INPUT (I)

The INPUT field describes the type of input device associated with the event (if any). This information is often redundant with that contained in the ACTION field. The standard input descriptors (and their default 2 letter abbreviations) are provided below.

KB	KEYBOARD
MS	MOUSE
TS	TOUCHSCREEN
JS	JOYSTICK
TB	TRACKBALL
SW	SWITCH
LP	LIGHT_POINTER

The standard input list is not meant to be exhaustive. Additional input devices can be defined in the logfile header as necessary.

METHOD (H)

The METHOD field describes the method used to select the current entry (if any). This information is often redundant with the ACTION and/or INPUT fields. The standard method descriptors (and their default 2 letter abbreviations) are provided below.

DR	DIRECT
SW	SWITCH
TS	TIMED_SWITCH
TM	TIMER
DW	DWELL
SC	SCANNING
UN	UNKNOWN

TYPE (P)

The TYPE of a logfile entry indicates the fundamental function of the user selection (or automatically generated entry). In a simple keyboard interface, the type of most entries will be "letter". In a scanning interface, many entries will be of type "advance to next group" and some will be of "select current group". In other interfaces, we might see types like "return to home page", "backspace", or "expand abbreviation". The type describes what's happening in a given entry.

We propose a hierarchical classification scheme for types. This scheme is not exhaustive, but should cover most of the types used for augmentative communication. As with the other logfile fields, additional type information may be provided in the header. We have divided all types into 12 separate categories. Each of these types is further divided into subtypes, and each of the subtypes divided further into "subsubtypes". For example, the letter "a" is type TEXT, subtype CHAR, and subsubtype LETTER while a "return to home page" action would be type CONTROL, subtype NAVIGATE, and subsubtype HOME. Since there are a large number of different types, the default descriptors (and abbreviations) are given in the Appendix.

TOKEN (K)

The TOKEN field contains a description of the function of the selected token: the "meaning" of the token. The TYPE field often provides information very similar to the TOKEN field, but the latter can provide additional detail. For example, the TYPE might indicate that a selection is a TEXT.CHAR.LETTER, while the TOKEN could indicate the particular letter (K:token.c or just K:c). Other examples might be K:token.dog for a picture of a dog, K:tk.shift for a shift key, and K:list.2 for the second item in a word prediction list. The TOKEN field may have spaces, but must be enclosed in quotes if so.

BUTTON (B)

The BUTTON field contains a description of the physical (or onscreen) button or key that precipitated the logfile entry (if any). For some single-page interfaces, this field may be equivalent with the TOKEN field. However, for most interfaces the meaning of a particular button can change over time. This is especially apparent in multi-page configurations, but may also be true for single-page interfaces with dynamic content.

The button descriptor can be of any form. As an example, in an interface that uses a grid-like arrangement of buttons (as many AAC devices do), the descriptor might indicate the row and column of the button: B:key.4.6 for the key in the 4th row and 6th column. The button descriptor may have spaces, but the field must be enclosed in quotes if so.

MESSAGE (M)

The MESSAGE is the written or symbolic message that is displayed on the token that is selected. It is often, but not always, the same as the OUTPUT. It may also be similar to the TOKEN or BUTTON field. If the message contains spaces, it must be enclosed in quotes.

CONTEXT (C)

The CONTEXT field provides information on the characters, word, or symbols that immediately precede the current logfile entry. This information must be enclosed in quotations. Context is useful for identifying selection strategies, computing prediction efficiency, and deriving a range of other useful measures. The context may be of variable length (i.e., as many characters, word, or symbols as is desired).

PAGE (G)

The PAGE field provides contextual information about the active page (folder, screen, etc.) at the time of the logfile entry. The name or code of the page should follow the field descriptor.

CENTER (N)

the CENTER field indicates the physical center of the selected token in x.y format. N:400.620 would be an example.

HEADER FORMAT

The logfile may contain an optional header that specifies the type and order of the fields in the logfile, as well as additional freeform information (for example, device used to create the logfile or a privacy statement). The header may be excluded from a file if (a) full field descriptors are provided for each logfile entry or (b) a canonical logfile format (for example, TIME and OUTPUT fields, as in the LAM) is used.

To ensure that the logging standard is as flexible and extendible as possible, the file header supports a number of different formats. In its most basic form, however, the header is extremely simple; It consists simply of a list of the fields that appear in each logfile entry. The order of the fields indicates the expected order of the fields in each logfile entry. Note that this is essential only for entries that do not contain explicit field specifiers.

Alternatively, we can specify a field *abbreviation* in the header. If a program opts to use field specifiers in each entry, we have already seen how a set of pre-established abbreviations can be used to shorten the entry length (for example, T instead of TIME). A program can specify an arbitrary abbreviation for each field using the following format. Arbitrary strings may be used for abbreviations, although we recommend the use of single- or dual-character abbreviations. The format for abbreviation specification is:

```
    <abbreviation>=<specifier> #Optional comment
```

Consider the following header example:

```
TIME
Z=TYPE #replace default of P with Z
OUTPUT
MS=MESSAGE #replace default of M with MS
```

This header specifies that each logfile entry consists of a TIME field (with default abbreviation of T), followed by a TYPE field (with new abbreviation of Z), followed by an OUTPUT field (with default abbreviation of O), followed by a MESSAGE field (with new abbreviation of MS). All of the following lines represent logfile entries that are compatible with this header information:

```
TIME: 04:12:21 TYPE: Letter OUTPUT: "b" MESSAGE: "b"
T: 04:12:21 Z: Letter O: "b" MS: "b"
MESSAGE: "b" T: 04:12:21 Z: Letter
04:12:21 Letter "b" "b"
```

In the first two examples, the field specifiers and field abbreviations are used, respectively. In the third example, both specifiers and abbreviations are intermixed, the order of the fields is changed, and the MESSAGE field is eliminated. In the last example, the field specifiers are implicit - the order of fields in the headers provides enough information to recover the individual fields.

The header can also be used to specify subfield equivalencies and to define new descriptors for fields that take descriptors. For example, we have described the default ACTION descriptors, such as MOUSECLICK, KEYDOWN, KEYUP, and AUTOSCAN. Like field specifiers, these descriptors also have default abbreviations (in this case MC, KD, KU, and AS). These can be replaced by arbitrary abbreviations using the following format:

```
A=ACTION
*M=MOUSECLICK
*KDOWN=KEYDOWN
*KUP=KEYUP
*S=AUTOSCAN
```

We have tried to define a set of ACTION descriptors that cover most available actions, but given the diversity of AAC technologies it would be impossible to specify an exhaustive list. Additional descriptors can therefore be defined in the header using the following format:

```
A=ACTION
*<abbreviation>=<descriptor> #Optional comment
```

For example, if an AAC device wanted to add an ACTION descriptor for a hypothetical input device called a "doubleflip button" the header entry might read as follows:

```
A=ACTION
*DB=DOUBLEFLIP #Doubleflip button press
```

In a logfile entry, fields of the form A:DOUBLECLICK and A:dB would then be acceptable (as would standard entries of the form A:MC and A:MOUSECLICK). An arbitrary number of descriptors can be added to any field that uses descriptors (ACTION, TYPE, DEVICE, INPUT, and METHOD). It makes no sense to add descriptors for fields such as TIME and OUTPUT that are not inherently limited to a small set of specific values.

Just as descriptors cannot be exhaustive, the list of fields supported by the logfile may also be insufficient for some devices. It is therefore possible to specify completely new fields using the standard field format:

```
    <abbreviation>=<specifier>
```

The header is terminated by a sequence of 3 dollar signs ($$$). This marker indicates that the logfile entries follow.

ENCRYPTION

An important aspect of the proposed logfile format is its support for message encryption to protect the privacy of the augmented communicator. When enabled, encryption applies only to the OUTPUT, MESSAGE, and CONTEXT fields. Three distinct types of encryption are supported. The algorithms for encrypting and decrypting character strings are simple enough to be easily programmed, but sophisticated enough to resist casual attempts at breaking the encryption code. Encryptions are based on an alphanumeric key of up to 60 characters. The key may be stored in the header of the logfile or may be omitted to provide even greater security. Note that even when the key appears in the logfile, it would be impossible to decrypt the contents without a standards-compliant analysis program such as ACQUA.

With PERFECT encryption all information about individual characters is discarded. All letters are replaced with X's such that the original strings cannot be recovered. Since spaces and punctuation marks are left unencrypted, it is still possible to analyze logfiles using PERFECT encryption for derived statistics such as communication rate and keystrokes per character. Of course, it is impossible to analyze any aspect of content.

With NORMAL encryption, letters are encrypted in a recoverable manner, but spaces and punctuations are not encrypted. It is therefore possible to analyze logfiles for derived statistics even if the encryption key is not available. Only with the proper key can content analysis be performed, however.

With STRONG encryption, all characters are encrypted (including spaces and punctuation marks). No meaningful analysis can be performed on such a logfile without the encryption key since no word or sentence delimiters are available.

A logfile can also be provided with a password. Both the encryption key and the password (assuming one is provided) are necessary to decrypt an encrypted file. The password must be defined (in encrypted form) in the logfile. The key may or may not be defined. Note that if either the key or unencrypted password are lost, it will not be possible to decrypt the logfile.

```
    key: <alphanumeric key>
    password: <encrypted alphanumeric password>
```

In January of 2004, an alternative encryption scheme was added to the logfile protocol. This scheme is indicated by a different key specification:

```
    key2: <alphanumeric key>
```

```
     password: <encrypted alphanumeric password>
```

Furthermore, in the new encryption scheme PERFECT encryption is indicated by the use of "key2p" rather than "key2".

Manufacturers interested in obtaining detailed information about the encryption algorithm should contact lesher@enkidu.net.

FOOTER FORMAT

A sequence of 3 dollar signs ($$$) at the beginning of a line indicates that there are no more entries in a logfile. Since all data after this point is ignored by any analysis programs, the manufacturer may choose to place arbitrary additional information in this footer. For example, our IMPACT software can be configured to compute rudimentary communication efficiency data to be stored in the footer.