Technical Note TN2045

AEBuild*, AEPrint* and Friends

CONTENTS

Introduction
AEBuilding Descriptor Records

- AEBuildDesc
- AEBuildError
- vAEBuildDesc

AEBuilding Apple Event Records

- AEBuildAppleEvent
- AEBuildParameters
- AEPrintDescToHandle

Descriptor-String Syntax

- Data Types
- Type Coercion

Complex Data Types

- Lists
- Records

Parameter Substitution
Using AEPrint* with gdb
Descriptor-String Grammar
Downloads

This Technote describes the AEBuild* suite of routines and the simple Apple event description language they accept. Also discussed is the AEPrint routine along with how it can be used inside of gdb.

The AEBuild* suite of routines provide simple to use and easy to maintain facilities for constructing complex Apple event structures in memory for sending information to other applications. AEPrint provides a symmetrical pretty printer routine for viewing complex Apple event structures as strings formatted using the same syntax as the strings AEBuild* is able to read.

This Note is directed at application developers who make extensive use of sophisticated Apple event structures in their applications.

[Mar 21 2002]

Introduction

The AEBuild* routines provide a very simple translation service for converting specially formatted strings into complex Apple event descriptors. Normally, creating complex Apple event descriptor records requires a large number of calls to the Apple event Manager routines to build up the descriptor piece by piece. The AEBuild* routines allow you to consolidate all of the calls required to construct a complex Apple event descriptor into a single system call that creates the desired structure as directed by a format string that you provide.

AEPrint provides a symmetrical pretty printer routine for displaying the contents of Apple event descriptor records. Strings created by AEPrint are of the same format as strings accepted by AEBuild. AEPrint can be very useful for viewing the contents of Apple event descriptor records when you are debugging your Apple event routines.

In many ways, the AEBuild* routines are very much like the standard C library's *printf suite of routines. The syntax for the 'format' string that you provide is very simple and allows for the substitution of data items into the Apple event descriptors being created. The remainder of this document describes the AEBuild* suite of routines and descriptor-string syntax that can be used with them.

Back to top

AEBuilding Descriptor Records

AEBuildDesc

AEBuildDesc provides a facility for compiling AEBuild descriptor-strings into Apple event descriptor records (AEDescs). Parameters are described below:

```
OSStatus AEBuildDesc(
    AEDesc* dst,
    AEBuildError* error,  /* can be NULL */
    const char* src,
    ...);
```

Parameters:

- `dst` - A pointer to an `AEDesc` record where the resulting descriptor should be stored.
- `error` - A pointer to a `AEBuildError` structure where additional information about any errors that occur will be saved. This is an optional parameter and the value `NULL` may be provided in its place if this information is not required.
- `src` - An `AEBuild` format string describing the `AEDesc` record to be created.
- `...` - A variable number of parameters as required by the format string provided in the `src` parameter.

Result:
A numeric result code indicating the success of the `AEBuildDesc` call. A value of `AEBuildSyntaxNoErr` (zero) means the call succeeded. The `error` parameter can be used to discover information about other errors.

Back to top

AEBuildError

`AEBuildError` defines a structure that can be passed to the `AEBuild*` routines to discover additional error information. The `AEBuild*` routines accept a pointer to this structure in an optional error parameter. While debugging a descriptor string you may wish to use this parameter to get more complete information about errors found in your descriptor-strings. The `AEBuildError` structure is declared as follows:

```
typedef UInt32 AEBuildErrorCode;

struct AEBuildError {
    AEBuildErrorCode fError;
    UInt32 fErrorPos;
};
typedef struct AEBuildError AEBuildError;
```

The purpose of this structure is to provide additional information about errors that occur during parsing of a descriptor-string. The `fError` field will contain one of the values shown in Table 1, and the `fErrorPos` field will contain the character position in the string where the error was noticed by the parser.

Table 1. Extended `AEBuild` error codes.

Constant Name	Value	Description
AEBuildSyntaxNoErr	0	(No error)
AEBuildSyntaxBadToken	1	Illegal character
AEBuildSyntaxBadEOF	2	Unexpected end of format string
AEBuildSyntaxNoEOF	3	Unexpected extra stuff past end
AEBuildSyntaxBadNegative	4	"-" not followed by digits
AEBuildSyntaxMissingQuote	5	Missing close "'"
AEBuildSyntaxBadHex	6	Non-digit in hex string
AEBuildSyntaxOddHex	7	Odd # of hex digits
AEBuildSyntaxNoCloseHex	8	Missing $ or "»"
AEBuildSyntaxUncoercedHex	9	Hex string must be coerced to a type
AEBuildSyntaxNoCloseString	10	Missing closing quote
AEBuildSyntaxBadDesc	11	Illegal descriptor

AEBuildSyntaxBadData	12	Bad data value inside (...)
AEBuildSyntaxNoCloseParen	13	Missing ")" after data value
AEBuildSyntaxNoCloseBracket	14	Expected "," or "]"
AEBuildSyntaxNoCloseBrace	15	Expected "," or "}"
AEBuildSyntaxNoKey	16	Missing keyword in record
AEBuildSyntaxNoColon	17	Missing ":" after keyword in record
AEBuildSyntaxCoercedList	18	Cannot coerce a list
AEBuildSyntaxUncoercedDoubleAt	19	"@@" substitution must be coerced

Back to top

vAEBuildDesc (varargs version)

The vAEBuildDesc routine allows you to encapsulate calls to AEBuildDesc in your own wrapper routines. You pass vAEBuildDesc a va_list reference to a previously defined, variable argument parameter list to use with the descriptor-string. The file <stdarg.h> defines macros for declaring and using the va_list data type. vAEBuildDesc provides the same functionality as AEBuildDesc.

```
OSStatus vAEBuildDesc(
    AEDesc* dst,
    AEBuildError* error,  /* can be NULL */
    const char* src,
    va_list args);
```

Parameters:

- dst - A pointer to an AEDesc record where the resulting descriptor should be stored.
- error - A pointer to a AEBuildError structure where additional information about any errors that occur will be saved. This is an optional parameter and the value NULL may be provided in its place if this information is not required.
- src - An AEBuild format string describing the AEDesc record to be created.
- args - A va_list value referencing the variable length argument list to be used by vAEBuildDesc.

Result:
A numeric result code indicating the success of the AEBuildDesc call. A value of AEBuildSyntaxNoErr (zero) means the call succeeded. The error parameter can be used to discover information about other errors.

All of the other AEBuild* routines that accept a variable length parameter list also include a include a varargs version with 'va_list args' in place of the variable length parameter list ('...'). For simplicity, we only mention these routines in following sections rather than providing complete definitions and descriptions, as they are the same as their corresponding variable argument equivalents.

Back to top

AEBuilding Apple Event Records

The syntax of the formatting string for an entire Apple event (as passed to AEBuildAppleEvent) is almost identical to that used to represent the contents of an Apple event record, without the curly braces. The event is defined as a sequence of name-value pairs, with optional parameters preceded with a tilde (~) character. The routine AEBuildAppleEvent can be used to build an entire Apple event record, and the routine AEBuildParameters can be used to add additional parameters to an existing Apple event record. These two routines are described in this section.

AEBuildAppleEvent

You can use the AEBuildAppleEvent routine to construct an entire Apple event record in a single call. It is very similar in function to the AECreateAppleEvent routine, except in addition to creating the AppleEvent record, it also constructs the parameters for the event from the last three arguments. For more information about the AECreateAppleEvent routine, see the Apple Event Manager documentation.

```
OSStatus AEBuildAppleEvent(
    AEEventClass theClass,
    AEEventID theID,
    DescType addressType,
    const void* addressData,
    long addressLength,
    short returnID,
    long transactionID,
    AppleEvent* result,
    AEBuildError* error,    /* can be NULL */
    const char* paramsFmt,
    ...);

NOTE: AEBuildAppleEvent has a varargs equivalent
      named vAEBuildAppleEvent.
```

Parameters:

- `theClass` - The event class for the resulting Apple event.
- `theID` - The event id for the resulting Apple event.
- `addressType` - The address type for the addressing information described in the next two parameters: usually one of `typeApplSignature`, `typeProcessSerialNumber`, or `typeKernelProcessID`.
- `addressData` - A pointer to the address information.
- `addressLength` - The number of bytes pointed to by the `addressData` parameter.
- `returnID` - Usually, set to the value `kAutoGenerateReturnID`. See the Apple Event Manager documentation for more information.
- `transactionID` - Usually, set to the value `kAnyTransactionID`. See the Apple Event Manager documentation for more information.
- `result` - A pointer to an `AEDesc` record where the resulting descriptor should be stored.
- `error` - A pointer to a `AEBuildError` structure where additional information about any errors that occur will be saved. This is an optional parameter and the value `NULL` may be provided in its place if this information is not required.
- `paramsFmt` - An `AEBuild` format string describing the `AppleEvent` record to be created.
- `...` - A variable number of parameters as required by the format string provided in the `paramsFmt` parameter.

Result:
A numeric result code indicating the success of the AEBuildDesc call. A value of `AEBuildSyntaxNoErr` (zero) means the call succeeded. The `error` parameter can be used to discover information about other errors.

IMPORTANT:
The identifier for the direct parameter in an Apple event record is four minus signs `'----'`. The minus sign has special meaning in `AEBuild` strings, and it should always be enclosed in single quotes when it is used to identify the direct parameter for an Apple event in a descriptor string.

Listing 1 provides an example of how you can use the `AEBuildAppleEvent` routine to create an Open Documents Apple event. The event created in the sample targets the Finder application, using its creator code (`'MACS'`).

Listing 1. An example of how to create an Open Documents Apple event using the `AEBuildAppleEvent` routine.

```
AliasHandle first_file, second_file;
const OSType finderSignature = 'MACS';
```

```
AppleEvent event;
OSErr err;
FSRef file1ref;
FSSpec file2spec;

    /* Construct the aliases...*/
err = FSNewAlias(NULL, &file1ref, &first_file);
if (err == noErr) {

  err = NewAlias(NULL, &file2spec, &second_file);
  if (err == noErr) {

    err = AEBuildAppleEvent(
        kCoreEventClass, kAEOpenDocuments,
        typeApplSignature, &finderSignagure, sizeof(finderSignature),
        kAutoGenerateReturnID, kAnyTransactionID,
        &event,   /* event to be created */
        NULL,   /* no error information required */
        "'----':[alis(@@), alis(@@)]", /* format string */
        first_file, /* param for 1st @@ */
        second_file); /* param for 2nd @@ */
```

Back to top

AEBuildParameters

AEBuildParameters can be called one or more times to add additional parameters or attributes to an existing Apple event record. The Apple event record should already have been created through either a call to AECreateAppleEvent or AEBuildAppleEvent. For more information about the AECreateAppleEvent routine, see the Apple Event Manager documentation.

```
OSStatus AEBuildParameters(
    AppleEvent* event,
    AEBuildError* error, /* can be NULL */
    const char* format,
    ...);

NOTE: AEBuildParameters has a varargs equivalent
      named vAEBuildParameters.
```

Parameters:

- event - A pointer to an AppleEvent record where the new parameters should be added.
- error - A pointer to a AEBuildError structure where additional information about any errors that occur will be saved. This is an optional parameter and the value NULL may be provided in its place if this information is not required.
- format - An AEBuild format string describing the AppleEvent parameters to be added.
- ... - A variable number of parameters as required by the format string provided in the format parameter.

Result:
A numeric result code indicating the success of the AEBuildParameters call. A value of AEBuildSyntaxNoErr (zero) means the call succeeded. The error parameter can be used to discover information about other errors.

Back to top

AEPrintDescToHandle

AEPrintDescToHandle provides a pretty printer facility for Apple event descriptor records. Information describing an AEDesc record is returned formatted in the special AEBuild* syntax. This facility is especially useful for looking at the contents of Apple event records sent by other applications and for debugging the Apple event descriptors created by your own application. Here is the definition for AEPrintDescToHandle:

```
OSStatus AEPrintDescToHandle(
    const AEDesc* desc,
    Handle* result);
```

Parameters:

- desc - A pointer to the Apple event descriptor record that should be printed out.
- result - A pointer to a location where a newly created Memory Manager 'Handle' containing the descriptor-string should be stored.

Result:
A numeric result code indicating the success of the AEPrintDescToHandle call. A value of AEBuildSyntaxNoErr (zero) means the call succeeded.

When AEPrintDescToHandle is asked to print an AEDesc, an AERecord, or an AEDescList, then the format of the printed output will match the input expected by AEBuildDesc. When printing, AEPrintDescToHandle tries to identify AERecords that have been coerced to different types and prints them as coerced records. Other structures that cannot be identified are dumped as hexadecimal data. For example, here is the AEPrintDescToHandle output for a list of three items:

```
[ "Mac OS X", 'null'(), 44]
```

AppleEvent records, though, are printed in a slightly different format. Here, the event class and event ID are printed at the beginning of a string, the parameter list is printed as a record in curly braces, and attributes are printed with their identifiers preceded by ampersand characters. For example, here is AEPrintDescToHandle output for an Open Documents Apple Event:

```
aevt\odoc{ ----:"Mac OS X",
           &addr:psn ($0000000000040001$),
           &subj:'null'(),
           &csig:magn($00010000$) }
```

The printed output produced for AppleEvent records is different than they syntax accepted by the AEBuild* suite of routines and this output cannot be used as input to AEBuildAppleEvent.

Back to top

Descriptor-String Syntax

Descriptor-strings are provided as null terminated c-style strings. Older instantiations of this library used some special MacRoman characters for some language symbols. These are still supported but new 7-bit ASCII alternatives have been added in addition to these older characters. Forward moving code should use the new preferred 7-bit ASCII characters.

Data Types

Four basic data types are provided in the descriptor-string syntax: integer, four-letter type codes, text strings, and hexadecimal data. Table 2 shows some examples of these types as expressed in the language.

Table 2. Basic AEBuild data types.

Type	Examples	Type code of AEDesc created	Description
Integer	1234 -5678	'shor' or 'long'	A sequence of decimal digits optionally preceded by a minus sign. Integers that are between -32768 and 32767 are converted to descriptors of type typeShortInteger. Values outside of that range are converted to typeLongInteger. If your implementation requires specific numeric types, you should always specify coercion.
Type Codes	whos longint 'long' m	'enum' (use coercion to change to 'type')	A type code must begin with a letter and is followed by any number of non-AEBuild-syntax characters. Only the first four characters are used: the type code will be truncated or padded with spaces to create a four character code. If enclosed in single quotes, then it may contain special AEBuild-syntax characters. Definitions for many of the type codes used by applications and system software can be found in the system header files: <AEDataModel.h>, <AERegistry.h>, and <AppleEvents.h>.
String	"A String" "Multiple lines are okay."	'TEXT'	Any sequence of characters between double quotes. The created 'TEXT' descriptor record will not include a terminating null character.
Hex Data	$4170706C65$ $ 0102 03ff e b 6 c $	?? (must be coerced to some type)	An even number of hex digits between dollar signs. Whitespace is ignored. Hex data has no inherent type. As a result, it must be coerced to some type whenever it is used.

IMPORTANT:
Watch out for type codes that contain special characters like commas, parentheses, braces, or non-trailing spaces, or that begin with a special character like '-'. These characters are used by AEBuild as part of its syntax. If you need to use any of these special characters in a type code, then enclose the type code in single quotes.

Back to top

Type Coercion

Any of the basic data types shown in Table 2 (except for hex data) has its own inherent data type associated with it that defines the type of the descriptor record that is created. If you would like a descriptor of a different type containing the same data, you can direct AEBuild* to create it by using the coercion syntax. The components of a coercion are the desired type code and the basic data type you would like to use. These items are provided in the following format (the desired type code followed by the basic data type enclosed in parentheses):

```
<desired type code> ( <data of any type> )
```

Directing AEBuild to perform a type coercion of this kind does not call any installed coercion handlers (the exception being the numeric coercions). Rather, it directs AEBuild to create a descriptor record containing exactly the same data but with a different type. Here are some examples:

```
sing(123)
type(line)
hexd("a string")
'blob'($4170706C65$)
'utxt'($0048 0045 004C  004C 004F$)
```

Installed Apple event coercion handlers are only called for numeric types (this is the only case where an

errAECoercionFail error can be returned). All other AEBuild coercions do nothing more than set the descriptor type field in the resulting descriptor record.

You can use coercion to create empty (or 'null') descriptor records by using one of the following syntax forms. All of these create empty descriptor records with zero length data.

```
null()
'null'()
()
```

The last form, (), though less explicit, can be used to create an empty or null descriptor.

Back to top

Complex Data Types

Apple event descriptors may contain two types of complex data structures containing, potentially, many other Apple event descriptor records. Lists, as the name suggests, contain a list of descriptor records. Records, contain a group of name-value pairs where the names are four letter type codes and the values are descriptor records. Both of these complex types are enclosed in Apple event descriptor records and so they may be used in a recursive fashion. That is to say, lists may contain lists or records and records may contain records or lists. The next two sections describe AEBuild declarations for these types in greater detail.

Lists

Apple event list descriptors contain zero or more Apple event descriptors. There is no requirement that they all be of the same type or even of the same format. In AEBuild descriptor-strings, a list of descriptors is specified by providing a, possibly empty, list of comma separated descriptor records enclosed in square brackets.

```
[ <descriptor>, <descriptor>... ]
```

This syntax will create a AEDescList descriptor record (an AEDesc record with the type 'list'). Here are some examples of valid AEBuild descriptor-strings defining lists:

```
[123, -456, "et cetera"]
[sing(1234), long(CODE)]
[["wheels", "within wheels"]]
[sing(1234), long(CODE), [[123, -456, "et cetera"], "within wheels"]]
[]
```

It is not possible to coerce a list to any other type - the descriptor type of a list is always set to 'list'. AEBuild descriptor-strings that include attempts to coerce a list to another type will not work.

Back to top

Records

A descriptor record is a group of name-value pairs in no particular order. In each name-value pair, the name is represented as a four letter type code and the value can be any valid descriptor. In AEBuild descriptor-string syntax an AERecord is declared as a comma separated list of name-value pairs enclosed in curly brackets:

```
{ <name> : <value>, <name> : <value>... }
```

Names and associated values are separated by ':' colon characters. By default, a record's type is set to 'reco', but a

record can be coerced to any type by preceding it's definition with the type code that should be used for the record:

```
<type code> { <name> : <value>, <name> : <value>... }
```

Here are some examples of AEBuild descriptor-strings containing valid AERecord declarations:

```
{x: 100, y:-100}
{'origin': {x: 100, y:-100}, extent: {x: 500, y:500},
   cont: [1, 5, 25]}
{}
rang{ star: 5, stop: 6}
```

The default type of a record is 'reco'. You can coerce a record structure to any type by preceding it with a type code. For example:

```
rang{ star: 5, stop: 6}
```

It is common for AERecords to be coerced to another type such as 'indx' or 'whos', but you should avoid coercing AERecords to common data types such as 'long' or 'TEXT' as that will confuse other applications and even facilities like AEPrint.

Back to top

Parameter Substitution

The AEBuild* suite of routines provides a powerful set of two substitution operators that can be used to read optional arguments provided to the AEBuild* routines. Using a method very similar to the facility provided by the standard-C library's printf routine, all of the AEBuild* routines accept an AEBuild descriptor-string together with a variable length parameter list. Arguments in the variable length parameter list are incorporated into the resulting descriptor record according to the placement of substitution operators in the AEBuild descriptor-string.

Substitution operators may be placed in an AEBuild descriptor-string in any place where a descriptor could be defined. The type of value created (and the type of parameter expected) depends on the context of the substitution operator in the AEBuild descriptor-string. Normally, the coercion operator applied to the substitution operator will determine the type of the descriptor created and the type of data expected as a command line parameter.

The special substitution operators are '@' and '@@'. Table 3 details how these operators are interpreted by AEBuild.

Table 3. Coercions and argument type requirements.

Type Coercion Specified	Type of parameter expected	Example	Comments
No coercion	AEDesc*	AEDesc mydesc; AEBuild(..., "@", &mydesc);	A plain '@' will be replaced with a descriptor parameter. '@@' cannot be specified in this case.
Numeric (bool, shor, long, sing, doub, exte)	short, short, long, float, short double, double	AEBuild([....] "long(@)", 44);	numeric types are provided 'as is' on the command line.
TEXT	char*	AEBuild([....] "TEXT(@)", "hello world");	A pointer to a null-terminated C string. The 'TEXT(@)' will be replaced with a descriptor of type 'TEXT' containing all of the text (except for the terminating zero byte) from the C-string. '@@' cannot be specified in this case.

Any other type	either a (length, pointer) pair or a Handle.	MyType myVar; AEBuild([....] "myst(@)", sizeof(myVar), &myVar);	Data for '@' will be read from two parameters: a long integer value specifying the number of bytes followed by a pointer to the bytes. If '@@' is provided, the data will be read from a Carbon Memory Manager 'Handle' value provided as a parameter.

Back to top

Using AEPrint* with gdb

When you are writing event handlers and AppleScript scripts, it is often useful to know the format of the Apple events that are sent between applications. This section shows how you can use the AEPrintDescToHandle routine in gdb to view the format of the events sent from a Mac OS X application.

gdb provides a 'call' facility that allows you to call a routine. In this example, the script shown in Listing 2 is used to call AEPrintDescToHandle to pretty print AppleEvent records to gdb's terminal. The example presented herein assumes that this script is saved in a file named 'gdb-aedesc'; but, if you would like to have this script loaded automatically every time you start gdb, then you can save it in your ~/.gdbinit file.

Listing 2. A gdb script for pretty printing AEDesc records.

```
define aedesc
  call (void *) malloc(4)
  set $aed_malloc=$
  call (long) AEPrintDescToHandle($arg0, $aed_malloc)
  if $ == 0
    printf "desc @ %p = {\n    type = '%.4s'\n    storage (%p) = %s\n}\n", \
      $arg0, $arg0, ((long *) $arg0)[1], **(char ***) $aed_malloc
    call (void) DisposeHandle(*(char ***) $aed_malloc)
  else
    printf "aedesc failed: error %d.\n", $
  end
  call (void) free($aed_malloc)
end
```

Let's assume we are running the Script Editor in Mac OS X and we have typed the script shown in Listing 3 into one of its windows, checked the syntax a few times, and run it a few times to verify that it is working correctly. And, after running it a few times we would like to find out more information about the Apple events that are being sent when this script runs.

Listing 3. A simple script.

```
tell application "Finder"
    activate
    open "Mac OS X"
end tell
```

The log of a terminal session shown in Listing 4 illustrates how we can use the aedesc gdb script to find out more information about the Apple events that the Script Editor is sending when we run the script. Comments that have been added to the script are preceded with '##' characters.

Listing 4. gdb session log.

```
    ## in a terminal window, start up gdb
[neithermac:/] apple% gdb
GNU gdb 5.0-20001113 (Apple version gdb-203) ([....] GMT 2001) (UI_OUT)
Copyright 2000 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you
are welcome to change it and/or distribute copies of it under certain
```

```
conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "powerpc-apple-macos10".


    ## read in our aedesc script.  I usually type 'source ' then drag
    ## and drop the file icon for the script into the terminal window,
    ## but you can also add the script to your gdb configuration
    ## (see 'man gdb').  the script defines the command 'aedesc' that
    ## we can use to call AEPrintDescToHandle to pretty print descriptors.
(gdb) source /Users/apple/gdb-aedesc

    ## next we attach to the Script Editor's process.  It has process
    ## id 1312 that we established before our gdb session.  you can
    ## look up the process id for running process using 'top -l'
    ## or 'ps -aux'
(gdb) attach 1312
Reading symbols for shared libraries . done
Reading symbols for shared libraries ..............................
.................. done
[Switching to process 1312 thread 0x1603]

    ## then, set a breakpoint on AESend so we can trap outgoing events
(gdb) break AESend
Breakpoint 1 at 0x731ddbec

    ## and tell gdb to allow the Script Editor to continue running.
(gdb) continue
Continuing.

    ## now, with the Script Editor we switch back and click on the
    ## 'run' button to run the script.


    ## we break on the the activate event
Breakpoint 1, 0x731ddbec in AESend ()

    ## ask the script to pretty print our descriptor (in $r3)
(gdb) aedesc $r3
$1 = (void *) 0x1bb8310
$2 = 0
desc @ 0xbfffe398 = {
    type = 'aevt'
    storage (0x150d858) = misc\actv{ &addr:psn ($0000000000040001$),
                &subj:'null'(), &csig:magn($00010000$) }
}

    ## and tell gdb to allow the Script Editor to continue running.
(gdb) continue
Continuing.

    ## we break on the the open document event
Breakpoint 1, 0x731ddbec in AESend ()

    ## ask the script to pretty print our descriptor (in $r3)
(gdb) aedesc $r3
$3 = (void *) 0x16ea1c0
$4 = 0
desc @ 0xbfffe2c8 = {
    type = 'aevt'
    storage (0x150d858) = aevt\odoc{ ----:"Mac OS X",
                &addr:psn ($0000000000040001$), &subj:'null'(),
                &csig:magn($00010000$) }
}

    ## and tell gdb to allow the Script Editor to continue running.
(gdb) continue
Continuing.
```

This same technique can be used to look at Apple events being sent by any Mac OS X application.

Back to top

Descriptor-String Grammar

What follows is a Backus Naur Form (BNF) grammar definition for AEBuild descriptor-strings.

```
# NOTE: comments inserted amongst rules are preceded by '#' characters.
# They are not part of the BNF.

# syntax rules for AEBuildAppleEvent
AEBuild-apple-event-expression ::= <event-keyword-list>

# event-keyword-list - list of zero or more event-keyword-pairs
# separated by commas
event-keyword-list ::= <event-keyword-pair> , <event-keyword-list>
event-keyword-list ::= <event-keyword-list>
event-keyword-list ::=

# event-keyword-pair an identifier object pair - preceded
# by a tilde ~ to indicate an optional parameter
event-keyword-pair ::= ~ <identifier> : <object>
event-keyword-pair ::= <identifier> : <object>


#syntax rules for AEBuildDesc
AEBuild-expression ::= <object>

object ::= <data>                   # Single AEDesc; shortcut for (data)
object ::= <structure>              # un-coerced structure
object ::= <identifier> <structure>  # coerced to some other type

structure ::= ( <data> )            # Single AEDesc
structure ::= [ <object-list> ]     # AEDescList type
structure ::= { <keyword-list> }    # AERecord type

# comma separated list of zero or more objects
object-list ::= <object-list> , object>
object-list ::= <object>
object-list ::=

# comma separated list of zero or more keyword/value pairs
keyword-list ::= <keyword-list> , <keyword-pair>
keyword-list ::= <keyword-pair>
keyword-list ::=
keyword-pair ::= <identifier> : <object>  # keyword/value pair


# @ and @@ are special tokens used for reading AEBuild
# parameters in to descriptors as they are constructed
data ::= @            # read data from AEBuild parameter
data ::= @@           # read data from AEBuild Handle parameter
data ::= <integer>    # 'shor' or 'long' unless coerced
data ::= <identifier>  # a 4-char type code ('type') unless coerced
data ::= <string>     # unterminated text: 'TEXT' type unless coerced
data ::= <hex-string>  # raw hex data; must be coerced to some type!
data ::=              # empty null data, useful for 'null()' coercions

integer ::= - <number>
integer ::= <number>
number ::= <number> <digit>
number ::= <digit>
digit ::= 0 .. 9

# no spaces allowed inside of identifers unless they
# are enclosed in single quotes.  identifiers
# are always padded (with spaces) or truncated to
# exactly 4 characters by AEBuild and friends.
identifier ::= <first-ident-letter> <ident-letter-list>
identifier ::= ' <any-letter letter-list> '   # straight quotes (preferred)
```

```
identifier ::= ‘ <any-letter letter-list> ‘   # curly quotes
ident-letter-list ::= <ident-letter-list> <ident-letter>
ident-letter-list ::= <ident-letter>
ident-letter-list ::=
first-ident-letter ::= alphabetic characters

# special AEBuild language characters cannot be included inside
# of identifiers.  To use identifiers containing them, enclose
# the identifer in single quotes.
ident-letter ::= any printable character excluding spaces
                 and special AEBuild characters

string ::= " <letter-list> "     # straight double quotes (preferred)
string ::= “ <letter-list> ”     # curly double quotes
letter-list ::= <letter-list> <any-letter>
letter-list ::= <any-letter>
letter-list ::=
any-letter ::= any printable character including spaces
               and special AEBuild characters.

# hex-strings may contain white space between digits
# so feel free to add them for formatting
hex-string ::= $ <hex-digit-list> $   # dollarsign quotes (preferred)
hex-string ::= « <hex-digit-list> »   # french quotes
hex-digit-list ::= <hex-digit-list> <hex-digit-pair>
hex-digit-list ::= <hex-digit-pair>
hex-digit-list ::=
hex-digit-pair ::= <hex-digit> <hex-digit>
hex-digit ::= digits 0 .. 9 or letters A .. F (case insensitive)
```

Back to top

Downloadables

Acrobat version of this Note (64K) Download

Back to top
