Technical Note TN2046

AEStream and Friends

CONTENTS

This Technote describes the `AEStream*` collection of APIs that can be used to construct Apple event records and Apple event descriptors using stream oriented calling conventions. These APIs allow you to use interesting and easy to maintain stream oriented algorithms for creating complex Apple event descriptor records.

The `AEStream*` routines are not intended to be a replacement for the other documented Apple Event Manager APIs. Rather, they supplement the other APIs and accomodate different styles of programming. Together with the other collection of Apple Event Manager APIs, the `AEStream*` routines make it simpler to create Apple event descriptor records in a wider variety of algorithms.

This Note is directed at application developers interested in using stream oriented algorithms for constructing complex Apple event descriptors.

[Mar 29 2002]

Introduction

The `AEStream*` routines allow you to append data to Apple event descriptor records in a way that is similar to opening and writing data to a file. Once an `AEStreamRef` has been opened, callers can append information to the stream using `AEStream*` routines describing the resulting descriptor that should be created. The commands sent to the stream are saved in the `AEStreamRef` but they are not actually combined to create a descriptor record until the `AEStreamRef` is closed.

Listing 1. An example illustrating how to make a simple descriptor.

```
AEStreamRef ref;
char* p = "Hello World";

    /* open the stream */
ref = AEStreamOpen();
```

```
    if (ref != NULL) {

        /* save a descriptor to the stream */
        err = AEStreamWriteDesc( ref, typeChar, p, strlen(p));
        if (err == noErr) {

            /* close the stream gathering the contents
            into theNewDesc, a new descriptor */
            AEDesc theNewDesc;
            err = AEStreamClose( ref, &theNewDesc);
            if (err == noErr) {

                /* here we would use the new descriptor
                for some purpose */

                /* dispose of the descriptor once we're
                finished with it. */
                AEDisposeDesc(&theNewDesc);
            }
        } else {
            /* if an error occurred creating the descriptor,
            close the stream and discard the result. */
            AEStreamClose( ref, NULL);
        }
    }
```

In the simplest case, an `AEStreamRef` can be used to create simple descriptor records as shown in Listing 1. But, going by this sample, it's not obvious that there is any advantage to using the `AEStream*` calls rather than using a simple call to `AECreateDesc` as shown in Listing 2. And, it is true, in this particular case, there is no particular advantage to using the `AEStream*` calls. However, in some similar circumstances the `AEStream*` calls may prove to be a better choice than simply calling `AECreateDesc`.

Listing 2. An example illustrating how to make a simple descriptor using `AECreateDesc`.

```
char* p = "Hello World";
err = AECreateDesc(typeChar, p, strlen(p), &textDesc);
```

Say, for example, the contents of the descriptor record are complex and can not be determined all at once, but you would like to accumulate that data in sequence as the contents of the descriptor record. To create such a descriptor using `AECreateDesc`, you would have to accumulate the data yourself, collect it into a large contiguous block of memory, and then copy the data to a descriptor by calling `AECreateDesc`. However, the `AEStream*` routines provide facilities for performing these operations without having to maintain any additional house keeping information. As shown in Listing 3, repeated calls to the `AEStreamWriteData` routine can be used to build-up the contents of a descriptor record. Also, when calling `AEStreamWriteData`, there is no need to specify all of the information that will be included in the descriptor at the same time (it could be collected across a number of calls in many different routines, for example).

Listing 3. An example illustrating how to build a simple descriptor, one letter at a time.

```
/* AddTextDesc adds a text descriptor containing the
text in the string pointed to by theText to an AEStreamRef.
It adds the text one character at a time illustrate how the
AEStream* routines can be used to build up the data contents
of descriptors incrementally. */
```

```
OSStatus AddTextDesc(AEStreamRef ref, char* theText) {
    char* p;
    OSStatus err;


        /* start a descriptor of type text */
    err = AEStreamOpenDesc( ref, typeChar);
    if (err == noErr) {

            /* add the string to the descriptor,
            one letter at a time */
        for (p=theText; *p; p++) {

                /* write a single character */
            err = AEStreamWriteData( ref, p, 1);
            if (err != noErr) break;
        }
        if (err == noErr) {

                /* mark the end of the descriptor */
            err = AEStreamCloseDesc(AEStreamRef ref);
        }
    }
    return err;
}

....

    AEStreamRef ref;
    ref = AEStreamOpen();
    if (ref != NULL) {

        /* call our AddTextDesc to add a text descriptor to
        the AEStreamRef */
    err = AddTextDesc(ref, "Hello World");
    if (err == noErr) {
        AEDesc theNewDesc;

            /* close the stream gathering the contents
            into theNewDesc, a new descriptor */
        err = AEStreamClose( ref, &theNewDesc);
        if (err == noErr) {

            /* here we would use the new descriptor
            for some purpose */

                /* dispose of the descriptor once we're

                finished with it. */
            AEDisposeDesc(&theNewDesc);
          }
      } else {
            /* if an error occurred creating the descriptor,
            close the stream and discard the result. */
        AEStreamClose( ref, NULL);
      }
  }
```

In the implementation of some algorithms, it may make more sense to incrementally build a descriptor record using techiques

similar to those used in Listing 3. Similarly, you may use the `AEStream*` routines to incrementally build `AEDescList` structures and `AERecord` structures. For instance, to build a list `AEDescList`, we simply bracket a sequence of calls writing individual descriptors with calls to `AEStreamOpenList` and `AEStreamCloseList`. The `AddAListOfStrings` routine defined in Listing 4 shows an example of how to use these routines to create a simple list of text descriptors. And, as one may expect, it is perfectly acceptable to nest calls to `AEStreamOpenList` and `AEStreamCloseList` to create lists inside of lists.

Listing 4. An example illustrating how to make a complex descriptor containing a list of strings.

```
    /* AddAListOfStrings appends a single AEDescList structure to
    the AEStreamRef containing a list of n descriptors of typeChar
    built using the array of string pointers provided in the strings
    parameter. */
OSStatus AddAListOfStrings(AEStreamRef ref, char** strings, long n) {
OSStatus err;
long i;

        /* start collecting items into a list */
err = AEStreamOpenList(ref);
if (err == noErr) {

        /* add all of our strings to the list
        using the AddTextDesc routine defined in
        Listing 3 */
    for (i=0; i<n; i++) {
        err = AddTextDesc(ref, strings[i]);
        if (err != noErr) break;

    }

        /* close the list */
    if (err == noErr) {
        err = AEStreamCloseList(ref);
    }
}
return err;
}

        /* a list of strings we will use to create our
        descriptor list */
    char* gStringList[] = {
        "Hello World",
        "Apple events",
        "AEStream*",
        "one last string"
    };

    AEStreamRef ref;
    long i;
    OSStatus err;

        /* open a stream */
    ref = AEStreamOpen();
    if (ref != NULL) {

        /* add a descriptor containing list of strings to
        the AEStream */
    err = AddAListOfStrings(ref, gStringList,
                        sizeof(gStringList)/sizeof(char*));

        /* close the AEStream, saving the new
```

```
        descriptor if no errors occurred */
    if (err == noErr) {
        AEDesc theNewDesc;

            /* close the stream gathering the contents
            into theNewDesc, a new descriptor */
        err = AEStreamClose( ref, &theNewDesc);
        if (err == noErr) {


            /* here we would use the new descriptor
            for some purpose */

                /* dispose of the descriptor once we're
                finished with it. */
                AEDisposeDesc(&theNewDesc);
            }
        } else {
                /* if an error occurred creating the descriptor,
                close the stream and discard the result. */
            AEStreamClose( ref, NULL);
        }
    }
```

Similar techniques can be used to construct complex `AERecord`s and to add parameters to Apple event records. For the most part, adding descriptors to records and parameters to Apple events proceeds in a similar fashion to creating a `AEDescList`; however, the `AEStream*` routines have additional provisions that allow you to specify record types and keywords associated with descriptors in records. Routines for creating records and adding elements to them are presented later in this document.

Back to top

AEStream* Tips

Here are some key points to remember when using the `AEStream*` routines:

- You create exactly one descriptor record between calls to `AEStreamOpen` and `AEStreamClose`. Don't expect `AEStream*` to automatically create a list for you if you write more than one descriptor between `AEStreamOpen` and `AEStreamClose`. If you want to create a list, then add the descriptors between calls to `AEStreamOpenList` and `AEStreamCloseList`.
- Be careful to balance all of your calls. For every call to `AEStreamOpenDesc` there must be a call to `AEStreamCloseDesc`, for every `AEStreamOpenList` call there must be a corresponding `AEStreamCloseList` call, and so on. As shown in the outline provided in Listing 5, it is possible make complex calling sequences to the `AEStream*` routines to create complex nested structures.

Listing 5. An outline of a calling sequence illustrating how calls to `AEStream*` can be nested to create a complex Apple event descriptor.

```
open a stream
   begin a list
      write a descriptor
      write a descriptor
      open a descriptor
         write some data
         write some data
```

```
      write some data
      write some data
   close the descriptor
   open a record
      set the record type
      write a key/descriptor pair
      write a key/descriptor pair
   close the record
   begin a list
      write a descriptor
      write a descriptor
      open a record
         set the record type
         write a key/descriptor pair
         write a key/descriptor pair
         close the record
      end a list
      write a descriptor
   end a list
close the stream -> a new descriptor
```

The `AEStream*` routines maintain state information that tracks the nesting of calls made to the library. This is necessary for its own operation, but it also uses this information to report errors. If your calls to the library are not balanced properly, `AEStreamClose` will return an `errAEStreamBadNesting` error when you try to close the `AEStreamRef`.

- Weigh other alternatives. Ask: will your code be simpler, more clear, and easier to maintain if you use the original Apple Event Manager routines such as `AECreateDesc` or the AEBuild* routines (see Technical Note TN2045) instead of the `AEStream*` routines? Using the `AEStream*` routines may improve the clarity of some implementations, but for others the `AEStream*` routines may not be the best choice.

- Be sure to dispose of the descriptor record created by `AEStreamClose`. It is your application's responsibility to dispose of the `AEDesc` record once it has finished using it.

Back to top

Opening and Closing an Apple event Stream

There are three routines that allow you to create and open a new `AEStreamRef` for collecting commands that will be used to create a new descriptor record when `AEStreamClose` is called. `AEStreamOpen` should be used for creating records of type `AEDesc`, `AEDescList`, and `AERecord`. The routine `AEStreamCreateEvent` can be used to create new `AppleEvent` records and the routine `AEStreamOpenEvent` can be used to open an existing `AppleEvent` record so you can add parameters to it. No matter which routine you choose to open an `AEStreamRef`, you must call `AEStreamClose` to close the stream and collect all of the commands issued to the `AEStreamRef` into the resulting descriptor record.

AEStreamOpen

```
AEStreamRef AEStreamOpen(void);
```

Result:
An `AEStreamRef` or, if an error occurs, the value `NULL`.

`AEStreamOpen` opens a new `AEStreamRef` that you can use for collecting commands describing an Apple event descriptor record. Once you have opened an `AEStreamRef` you can call the other `AEStream*` routines to describe the format of the descriptor record you would like to create. The example provided in Listing 1 shows how you would call this routine.

Back to top

AEStreamClose

```
OSStatus AEStreamClose(
    AEStreamRef ref,
    AEDesc* desc);
```

Parameters:

- `ref` – An `AEStreamRef` created by either `AEStreamOpen`, `AEStreamCreateEvent` or `AEStreamOpenEvent`.

- `desc` – either a pointer to `AEDesc` record where the sequence of items written to the `AEStreamRef` should be saved or the value `NULL`. If this parameter is set to NULL, then `AEStreamClose` will discard the result and dispose of the `AEStreamRef` (no matter what state it is in).

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

`AEStreamClose` closes and deallocates an `AEStreamRef` created by calling one of the routines `AEStreamOpen`, `AEStreamCreateEvent` or `AEStreamOpenEvent`. If a pointer to a descriptor is provided in the second parameter, then the resulting descriptor record will be stored in that location. If the `AEStreamRef` was created by `AEStreamOpenEvent` or `AEStreamCreateEvent`, then the resulting descriptor will contain a complete Apple event.

Note:
It is the calling application's responsibility to dispose of the descriptor record returned by `AEStreamClose`. Once you are finished with the descriptor record, you should dispose of it by calling `AEDisposeDesc`.

Providing a `NULL` pointer in the `desc` parameter instructs `AEStreamClose` to discard the result and dispose of the `AEStreamRef` (no matter what state it is in). When you call `AEStreamClose` in this way, you do not need to worry about balancing nested calls to `AEStream*` (such as, for example, `AEStreamOpenList` and `AEStreamCloseList`). This can be particularly useful in error handling situations when you would like to dispose of the `AEStreamRef`, but you do not necessarily know anything about what state it is in.

All nested sequences of calls must be appropriately balanced before `AEStreamClose` is called. That is to say, every `AEStreamOpenList` call must have a corresponding `AEStreamCloseList` call, every `AEStreamOpenRecord` call must have a corresponding `AEStreamCloseRecord` call, and so on. Calling `AEStreamClose` to close an `AEStreamRef` after nested calls have not been balanced properly will result in a `errAEStreamBadNesting` error. The example listing in Listing 1 illustrates how to call `AEStreamClose` to obtain the resulting descriptor record and how to call `AEStreamClose` discarding the result after an error has occurred.

In the case where `AEStreamClose` fails and returns an error, the `AEDesc` record pointed to by the `desc` parameter will be set to a descriptor of `typeNull`. Because of this, it is always safe to call `AEDisposeDesc` on the descriptor record returned by `AEStreamClose` no matter what result code is returned.

returned by `AEStreamClose` no matter what result code is returned.

Back to top

Writing Descriptors

The `AEStream*` routines include three different facilities that allow you to add individual descriptor records to an `AEStreamRef`.

0. `AEStreamWriteAEDesc`, allows you to write a pre-built `AEDesc` record (of any format) to an `AEStreamRef`. This allows you to take pre-assembled descriptors and build them into complex descriptor structures in much the same way as you would use constants in any program.

1. `AEStreamWriteDesc` lets you provide a data buffer and a `typeCode` that will be used to create the descriptor.

2. The `AEStreamOpenDesc`, `AEStreamWriteData`, and `AEStreamCloseDesc` calls allow you to create a descriptor record by building up the data it contains incrementally. All of the data provided across multiple calls to `AEStreamWriteData` is combined when the resulting descriptor record is created. The example shown in Listing 3 shows how to use these three calls to build a descriptor record incrementally.

These routines are described in this section. Keep in mind that all of these routines can be used to create individual descriptor records (`AEDesc` records). These same routines are used for adding descriptor records to `AEDescList` and `AERecord` structures, but to do so calls to these routines must be bracketed by calls to `AEStreamOpenList`/`AEStreamCloseList` and `AEStreamOpenRecord`/`AEStreamCloseRecord`, respectively.

AEStreamWriteAEDesc

```
OSStatus AEStreamWriteAEDesc(
    AEStreamRef ref,
    const AEDesc *desc);
```

Parameters:

- `ref` - An `AEStreamRef` created by either `AEStreamOpen`, `AEStreamCreateEvent` or `AEStreamOpenEvent`.

- `desc` - a pointer to an `AEDesc` record that should be copied into the `AEStreamRef`. `AEStreamWriteAEDesc` copies the `AEDesc` immediately so there is no need to retain this storage after calling `AEStreamWriteAEDesc`.

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

`AEStreamWriteAEDesc` copies an entire existing Apple event descriptor record to the `AEStreamRef`.

This routine useful in cases where you have a pre-assembled descriptor record that you would like to use in the descriptor you are creating. For example, say you have a complex object specifier record that could be costly to create every time you build a descriptor, but you would like to add it to many different descriptor records. In this case, it would be more efficient to create the object specifier record at program start up time and then use it, much like a constant, again and again, in calls to `AEStreamWriteAEDesc`.

Back to top

AEStreamWriteDesc

```
OSStatus AEStreamWriteDesc(
    AEStreamRef ref,
    DescType newType,
    const void* data,
    Size length);
```

Parameters:

- `ref` – An `AEStreamRef` created by either `AEStreamOpen`, `AEStreamCreateEvent` or `AEStreamOpenEvent`.
- `newType` – A type code for the new `AEDesc` being copied to the `AEStreamRef`.
- `data` – A pointer to a block of memory containing `length` bytes of data that will be used in the new `AEDesc` being copied to the `AEStreamRef`. `AEStreamWriteDesc` copies the data immediately so you do not need to retain this memory after calling `AEStreamWriteDesc`.
- `length` – The number of bytes pointed to by the data parameter.

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

`AEStreamWriteDesc` allows you to provide a `typeCode` and a data buffer that will be used to create a descriptor record. With this routine, you must provide all of the data that will be used in the descriptor record all at once. If you would like to provide the data incrementally, then you should use the `AEStreamOpenDesc`, `AEStreamWriteData`, and `AEStreamCloseDesc` calls (described next).

Back to top

AEStreamOpenDesc

```
OSStatus AEStreamOpenDesc(
    AEStreamRef ref,
    DescType newType);
```

Parameters:

- `ref` – An `AEStreamRef` created by either `AEStreamOpen`, `AEStreamCreateEvent` or `AEStreamOpenEvent`.
- `newType` – A type code for the new `AEDesc` being copied to the `AEStreamRef`.

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

Calling `AEStreamOpenDesc` marks the beginning of a sequence of zero or more calls to `AEStreamWriteData`. All of the calls to `AEStreamWriteData` made between calls to `AEStreamOpenDesc` and `AEStreamCloseDesc` will be combined to create a descriptor with the `typeCode` specified in the call to `AEStreamOpenDesc`. A call to `AEStreamOpenDesc` may be followed by any number of calls `AEStreamWriteData` and must be balanced with a call to `AEStreamCloseDesc`.

Back to top

AEStreamWriteData

```
OSStatus AEStreamWriteData(
    AEStreamRef ref,
    const void* data,
    Size length);
```

Parameters:

- `ref` – An `AEStreamRef` created by either `AEStreamOpen`, `AEStreamCreateEvent` or `AEStreamOpenEvent`. `AEStreamOpenDesc` or `AEStreamOpenKeyDesc` must have been called on this `AEStreamRef` prior to calling `AEStreamWriteData`.

- `data` – A pointer to a block of memory containing `length` bytes of data that will be used in the new `AEDesc` being copied to the `AEStreamRef`. `AEStreamWriteData` copies the data immediately so you do not need to retain this memory after calling `AEStreamWriteData`.

- `length` – The number of bytes pointed to by the data parameter.

Result:
A numeric result code indicating the success of the call. A value of `noErr`

(zero) means the call succeeded.

`AEStreamWriteData` appends data to the current descriptor record being defined in the `AEStreamRef`. You can call this routine any number of times to build up the data contents of a descriptor record incrementally. Calls to `AEStreamWriteData` must be preceded by either a call to `AEStreamOpenDesc` or a call to `AEStreamOpenKeyDesc`. After calling `AEStreamWriteData` one or more times to define the data contents of a descriptor record, you must call `AEStreamCloseDesc` to complete the descriptor's definition and balance the preceding call to either `AEStreamOpenDesc` or `AEStreamOpenKeyDesc`. The example shown in Listing 3 shows how to call `AEStreamWriteData`.

Back to top

AEStreamCloseDesc

```
OSStatus AEStreamCloseDesc(AEStreamRef ref);
```

Parameters:

- `ref` - An `AEStreamRef` created by either `AEStreamOpen`, `AEStreamCreateEvent` or `AEStreamOpenEvent`. `AEStreamOpenDesc` or `AEStreamOpenKeyDesc` must have been called on this `AEStreamRef` prior to calling `AEStreamCloseDesc`.

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

`AEStreamCloseDesc` is used to mark the end of a descriptor record being written to an `AEStreamRef`. After calling `AEStreamWriteData` one or more times to define the data contents of a descriptor record, you must call this routine to complete the descriptor's definition and balance the preceding call to either `AEStreamOpenDesc` or `AEStreamOpenKeyDesc`. The example shown in Listing 3 shows how to call `AEStreamCloseDesc`.

Back to top

Writing Lists

Routines in this section are used to delimit a group of descriptor records that will be combined to create at `AEDescList` structure in the resulting descriptor.

AEStreamOpenList

```
OSStatus AEStreamOpenList(AEStreamRef ref);
```

Parameters:

- `ref` - An `AEStreamRef` created by either `AEStreamOpen`, `AEStreamCreateEvent` or `AEStreamOpenEvent`.

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

Calling `AEStreamOpenList` marks the beginning of a sequence of zero or more descriptor definitions that will be combined to create a single `AEDescList` structure in the resulting descriptor record. Every call to `AEStreamOpenList` must be balanced with a corresponding call to `AEStreamCloseList`. The descriptors included in an `AEDescList` structure can be:

0. `AEDesc`s defined using any of the three methods described in the Writing Descriptors section.
1. `AEDescList`s defined using the routines described in this section.
2. `AERecord`s defined using the routines described in the Writing Records section.

Back to top

AEStreamCloseList

```
OSStatus AEStreamCloseList(AEStreamRef ref);
```

Parameters:

- `ref` – An `AEStreamRef` created by either `AEStreamOpen`, `AEStreamCreateEvent` or `AEStreamOpenEvent`. `AEStreamOpenList` must have been called on this `AEStreamRef` prior to calling `AEStreamCloseList`.

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

`AEStreamCloseList` is used to mark the end of a list started by calling to `AEStreamOpenList`. Call this routine after writing some descriptors to an `AEStreamRef` to balance the preceding call to `AEStreamOpenList`.

Back to top

Writing Records

Routines provided for writing `AERecord` structures are very similar to the set of routines provided for writing `AEDescList` structures. The main difference is the routines for writing `AERecord` structures include provisions for specifying type codes associated with `AERecord` and the keywords associated with record elements. For flexibility, the `AEStream*` routines provide more than one way to specify these additional elements.

AEStreamOpenRecord

```
OSStatus AEStreamOpenRecord(
    AEStreamRef ref,
    DescType newType);
```

Parameters:

- `ref` – An `AEStreamRef` created by either `AEStreamOpen`, `AEStreamCreateEvent` or `AEStreamOpenEvent`.
- `newType` – A type code for the new `AERecord` being copied to the `AEStreamRef`. Sometimes `typeAERecord` (`'reco'`), but usually a different type is used.

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

Calling `AEStreamOpenRecord` marks the beginning of a sequence of zero or more keyword/descriptor definitions that will be combined to create a single `AERecord` structure in the resulting descriptor record. Every call to `AEStreamOpenRecord` must be balanced with a corresponding call to `AEStreamCloseRecord`. Each keyword/descriptor definition is introduced with a call to either `AEStreamWriteKeyDesc`,

`AEStreamOpenKeyDesc`, or `AEStreamWriteKey`.

Back to top

AEStreamCloseRecord

```
OSStatus AEStreamCloseRecord(AEStreamRef ref);
```

Parameters:

- `ref` – An `AEStreamRef` created by either `AEStreamOpen`, `AEStreamCreateEvent` or `AEStreamOpenEvent`. `AEStreamOpenRecord` must have been called on this `AEStreamRef` prior to calling `AEStreamCloseRecord`.

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

`AEStreamCloseRecord` is used to mark the end of a record started by calling to `AEStreamOpenRecord`. Call this routine after writing some keyword/descriptor pairs to an `AEStreamRef` to balance the preceding call to `AEStreamOpenRecord`.

Back to top

AEStreamSetRecordType

```
OSStatus AEStreamSetRecordType(
    AEStreamRef ref,
    DescType newType);
```

Parameters:

- `ref` – An `AEStreamRef` created by either `AEStreamOpen`, `AEStreamCreateEvent` or `AEStreamOpenEvent`.

- `newType` – A type code for the new `AERecord` currently being copied to the `AEStreamRef`. Since `AEStreamOpenRecord`/`AEStreamCloseRecord` calls can be nested, this call will set the type for the record associated with the most recent `AEStreamOpenRecord` call.

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

`AEStreamSetRecordType` can be called after `AEStreamOpenRecord` to set the type of the `AERecord` being defined to a different type than the type specified in the `newType` parameter to the `AEStreamOpenRecord` call. In the case where your code is creating nested records, `AEStreamSetRecordType` will set the type of the `AERecord` associated with the most recent call to `AEStreamOpenRecord`. This routine can only be called between

AEStreamOpenRecord and AEStreamCloseRecord calls.

Back to top

AEStreamWriteKeyDesc

```
OSStatus AEStreamWriteKeyDesc(
    AEStreamRef ref,
    AEKeyword key,
    DescType newType,
    const void* data,
    Size length);
```

Parameters:

- `ref` – An `AEStreamRef` created by either `AEStreamOpen`, `AEStreamCreateEvent` or `AEStreamOpenEvent`. Since `AEStreamOpenRecord`/`AEStreamCloseRecord` calls can be nested, this call will add a new `AEKeyword`/`AEDesc` pair to the `AERecord` associated with the most recent `AEStreamOpenRecord` call.

- `key` – The `AEKeyword` associated with the new `AEDesc` being added to the current `AERecord`.

- `newType` – A type code for the new `AEDesc` being copied to the `AEStreamRef`.

- `data` – A pointer to a block of memory containing `length` bytes of data that will be used in the new `AEDesc` being copied to the `AEStreamRef`. `AEStreamWriteKeyDesc` copies the data immediately so you do not need to retain this memory after calling `AEStreamWriteKeyDesc`.

- `length` – The number of bytes pointed to by the data parameter.

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

Call `AEStreamWriteKeyDesc` to write a complete keyword/descriptor pair to an `AEStreamRef` for inclusion in a `AERecord`. This routine can only be called between `AEStreamOpenRecord` and `AEStreamCloseRecord` calls. `AEStreamWriteKeyDesc` is analogous to the Apple Event Manager routine `AEPutParamPtr`, and it is nearly the same as `AEStreamWriteDesc` (except it has an additional `AEKeyword` parameter). The sample shown in Listing 6 shows how you could call this routine.

In the case where your code is creating nested records, `AEStreamWriteKeyDesc` will add a `AEKeyword`/`AEDesc` pair to the `AERecord` associated with the most recent call to `AEStreamOpenRecord` or `AEStreamOpenEvent`. `AEStreamWriteKeyDesc` can only be called while an `AERecord` is being written – it cannot be called to add `AEKeyword`/`AEDesc` pairs to a `AERecord` while you are writing to a nested `AEDescList` inside of that `AERecord`.

Back to top

AEStreamOpenKeyDesc

```
OSStatus AEStreamOpenKeyDesc(
    AEStreamRef ref,
    AEKeyword key,
    DescType newType);
```

Parameters:

- `ref` – An `AEStreamRef` created by either `AEStreamOpen`, `AEStreamCreateEvent` or `AEStreamOpenEvent`. Since `AEStreamOpenRecord/AEStreamCloseRecord` calls can be nested, this call will add a new `AEKeyword/AEDesc` pair to the `AERecord` associated with the most recent `AEStreamOpenRecord` call.

- `key` – The `AEKeyword` associated with the new `AEKeyword/AEDesc` pair being added to the current `AERecord`.

- `newType` – A type code for the new `AEDesc` being copied to the `AEStreamRef`.

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

Call `AEStreamOpenKeyDesc` to start the definition of a keyword/descriptor pair for inclusion in an `AERecord`. `AEStreamOpenKeyDesc` is the same as `AEStreamOpenDesc` with the exception that it includes an `AEKeyword` parameter that will be used as the `AERecord` element's keyword. You should use this routine when you would like to provide the data associated with a keyword/descriptor pair in an incremental fashion with repeated calls to `AEStreamWriteData`. After calling `AEStreamOpenKeyDesc` you should call `AEStreamWriteData` a number of times to define the data contents for the descriptor before calling `AEStreamCloseDesc` to complete the definition.

In the case where your code is creating nested records, `AEStreamOpenKeyDesc` will begin a `AEKeyword/AEDesc` pair in the `AERecord` associated with the most recent call to `AEStreamOpenRecord` or `AEStreamOpenEvent`. `AEStreamOpenKeyDesc` can only be called while an `AERecord` is being written – it cannot be called to begin `AEKeyword/AEDesc` pairs in a `AERecord` while you are writing to a nested `AEDescList` inside of that `AERecord`.

Back to top

AEStreamWriteKey

```
OSStatus AEStreamWriteKey(
    AEStreamRef ref,
    AEKeyword key);
```

Parameters:

- `ref` – An `AEStreamRef` created by either `AEStreamOpen`, `AEStreamCreateEvent` or `AEStreamOpenEvent`.

- `key` – The `AEKeyword` associated with the new `AEKeyword/AEDesc` pair being added to the current `AERecord`.

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

Call `AEStreamWriteKey` to start the definition of a keyword/descriptor pair for inclusion in an `AERecord`. This routine only writes the keyword part of the definition and it must be followed by a sequence of `AEStream*` calls that define exactly one descriptor record. That descriptor can be any of the following:

- An `AEDesc` defined using any of the three methods described in the Writing Descriptors section.
- An `AEDescList` defined using the routines described in the Writing Lists section.
- An `AERecord` defined using the routines described in this section.

The sample shown in Listing 6 shows how you could call this routine.

In the case where your code is creating nested records, `AEStreamWriteKey` will begin a `AEKeyword/AEDesc` pair in the `AERecord` associated with the most recent call to `AEStreamOpenRecord` or `AEStreamOpenEvent`. `AEStreamWriteKey` can only be called while an `AERecord` is being written – it cannot be called to begin `AEKeyword/AEDesc` pairs in a `AERecord` while you are writing to a nested `AEDescList` inside of that `AERecord`.

Back to top

Writing Apple Event Records

The `AEStream*` routines can be used to create complete Apple event records. `AEStreamCreateEvent` can be used to define a new `AppleEvent` record, and the routine `AEStreamOpenEvent` can be used to supplement an existing one. Since the contents of `AppleEvent` records are formatted the same as `AERecords`, we use the same routines for filling in the parameters as we use for adding `AERecord` elements. For Apple events, we use the routines `AEStreamWriteKeyDesc`, `AEStreamOpenKeyDesc`, or `AEStreamWriteKey` defined in the Writing Records section. `AppleEvent` records include provisions for optional parameters. So that you can specify which of the parameters you supply are optional, `AEStream*` provides the `AEStreamOptionalParam` call.

AEStreamOpenEvent

```
AEStreamRef AEStreamOpenEvent(AppleEvent *event);
```

Parameters:

- `event` – An existing Apple event record that you would like to add parameters to.

Result:
An `AEStreamRef` or, if an error occurs, the value `NULL`.

`AEStreamOpenEvent` allows you to open an existing `AppleEvent` record in an `AEStreamRef` so you can use the `AEStream*` routines to append additional parameters to the event. Once you have opened an event record the contents of the event record are copied into the `AEStreamRef`. When you have finished describing the parameters for the event record, call `AEStreamClose` to save them to an `AppleEvent` record. The example shown in Listing 6 illustrates how to call this routine.

If there is not enough storage available to complete the operation, `AEStreamOpenEvent` will return `NULL` and the `AppleEvent` parameter will remain unchanged. Otherwise, if successful, `AEStreamOpenEvent` will return a vaild `AEStreamRef` and the `AppleEvent` parameter will be set to a descriptor of `typeNull`. When you open an

AppleEvent record using AEStreamOpenEvent, you must define the parameters using the same routines you would use for adding keyword/descriptor pairs to records as described in the Writing Records section together with the AEStreamOptionalParam routine described in this section.

Listing 6. An example illustrating how to call AEStreamOpenEvent.

```
AppleEvent event;
AEStreamRef ref;
OSStatus err;
char* p = "Hello World";
....
ref = AEStreamOpenEvent(&event);
if (ref != NULL) {

    /* add a direct parameter */
err = AEStreamWriteKeyDesc( ref, keyDirectObject, typeChar, p, strlen(p))
if (err == noErr) {

        /* add another, optional parameter. */
    err = AEStreamWriteKey( ref, 'mine');
    if (err == noErr) {

            /* add all of our strings to the list
            using the AddTextDesc routine defined in
            Listing 3 */
        err = AddTextDesc( ref, "this is an optional parameter");
        if (err == noErr) {

                /* flag the parameter with the keyword 'mine' as an
                optional parameter. */
            err = AEStreamOptionalParam( ref, 'mine');
            if (err == noErr) {

                /* close the stream */
                err = AEStreamClose(ref, &event);
                if (err == noErr) {

                    /* send the event */
                    err = AESend(&event, ...);
                ....
```

AEStreamOpenEvent clears the contents of the AppleEvent structure passed to it and sets it to a 'null' descriptor after it reads the AppleEvent into the AEStreamRef. To copy the AppleEvent back into this variable after you are finished writing parameters to it, call AEStreamClose providing this same structure as the destination for the resulting descriptor record.

Back to top

AEStreamCreateEvent

```
AEStreamRef AEStreamCreateEvent(
    AEEventClass clazz,
    AEEventID id,
    DescType targetType,
    const void* targetData
```

```
    const void* targetData,
    long targetLength,
    short returnID,
    long transactionID);
```

Parameters:

- `clazz` – The event class for the resulting Apple event.
- `id` – The event id for the resulting Apple event.
- `targetType` – The address type for the addressing information described in the next two parameters: usually one of `typeApplSignature`, `typeProcessSerialNumber`, or `typeKernelProcessID`.
- `targetData` – A pointer to the address information.
- `targetLength` – The number of bytes pointed to by the `targetData` 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:

An `AEStreamRef` or, if an error occurs, the value `NULL`.

`AEStreamCreateEvent` calls `AECreateAppleEvent` before calling `AEStreamOpenEvent` to open an `AEStreamRef` for adding parameters to the event.

Back to top

AEStreamOptionalParam

```
OSStatus AEStreamOptionalParam(
    AEStreamRef ref,
    AEKeyword key);
```

Parameters:

- `ref` – An `AEStreamRef` created by either `AEStreamOpen`, `AEStreamCreateEvent` or `AEStreamOpenEvent`.
- `key` – The `AEKeyword` associated with the any `AEKeyword/AEDesc` pair being added to an `AppleEvent`. `AEStreamCreateEvent` or `AEStreamOpenEvent` must precede any calls to `AEStreamOptionalParam`.

Result:

A numeric result code indicating the success of the call. A value of `noErr`

(zero) means the call succeeded.

The `AEStreamOptionalParam` allows you to designate optional parameters (by `AEKeyword`) in `AppleEvent` records. The sample shown in Listing 6 shows how you could call this routine.

Back to top

Downloadables

Acrobat version of this Note (104K) Download

Back to top