NAME

auparse_feed − feed data into parser

SYNOPSIS

#include <auparse.h>

int auparse_feed(auparse_state_t * *au*,**const***char***"** *data*,**size_t"** *data_len);"*

au The audit parse state

data a buffer of data to feed into the parser, it is *data_len* bytes long. The data is copied in the parser, upon return the caller may free or reuse the data buffer.

data_len
number of bytes in *data*

DESCRIPTION

auparse_feed supplies new data for the parser to consume. *aupar se_init()* must have been called with a source type of AUSOURCE_FEED and a NULL pointer.

The parser consumes as much data as it can invoking a user supplied callback specified with *auparse_add_callback* with a cb_event_type of *AUPARSE_CB_EVENT_READY* each time the parser recognizes a complete event in the data stream. Data not fully parsed will persist and be prepended to the next feed data. After all data has been feed to the parser *auparse_flush_feed* should be called to signal the end of input data and flush any pending parse data through the parsing system.

RETURN VALUE

Returns −1 if an error occurs; otherwise, 0 for success.

EXAMPLE

```
void
auparse_callback(auparse_state_t *au, auparse_cb_event_t cb_event_type,
                 void *user_data)
{
   int *event_cnt = (int *)user_data;

   if (cb_event_type == AUPARSE_CB_EVENT_READY) {
      if (auparse_first_record(au) <= 0) return;
      printf("event: %d\n", *event_cnt);
      printf("records:%d\n", auparse_get_num_records(au));
      do {
         printf("fields:%d\n", auparse_get_num_fields(au));
         printf("type=%d ", auparse_get_type(au));
         const au_event_t *e = auparse_get_timestamp(au);
         if (e == NULL) return;
         printf("event time: %lu.%u:%lu\n",
               (long unsigned)e−>sec, e−>milli, e−>serial);
         auparse_first_field(au);
         do {
            printf("%s=%s (%s)\n", auparse_get_field_name(au),
                  auparse_get_field_str(au),
                  auparse_interpret_field(au));
         } while (auparse_next_field(au) > 0);
         printf("\n");
```

```
        } while(auparse_next_record(au) > 0);
        (*event_cnt)++;
    }
}

main(int argc, char **argv)
{
    char *filename = argv[1];
    FILE *fp;
    char buf[256];
    size_t len;
    int *event_cnt = malloc(sizeof(int));

    au = auparse_init(AUSOURCE_FEED, 0);
    auparse_set_eoe_timeout(2);

    *event_cnt = 1;
    auparse_add_callback(au, auparse_callback, event_cnt, free);

    if ((fp = fopen(filename, "r")) == NULL) {
        fprintf(stderr, "could not open ’%s’, %s\n", filename, strerror(errno));
        return 1;
    }

    while ((len = fread(buf, 1, sizeof(buf), fp))) {
        auparse_feed(au, buf, len);
    }
    auparse_flush_feed(au);
    auparse_destroy(au);
}
```

SEE ALSO

auparse_add_callback(3), **auparse_flush_feed**(3), **auparse_feed_age_events**(3), **auparse_feed_has_data**(3), **auparse_metrics**(3)

AUTHOR

John Dennis