User Guide for Implementing NADF Adaptors

Abdelaziz Mounji
Institut d'Informatique, FUNDP
E-mail: amo@info.fundp.ac.be

January 25, 1995

1 Introduction

The purpose of this paper is to specify the generic audit record format used by ASAX. It also provides guidelines for implementing programs that convert a native file to a NADF format. Such a converter program is called a *format adaptor*.

2 Why a Common Format ?

ASAX is a universal tool for data stream analysis (and in particular a security audit trail analysis). That means ASAX is theoretically able to analyse arbitrary sequential files. This is achieved by translating the native file to a universal format called *Normalized Audit Data Format*. This ensures target system independence and avoids the need to tune ASAX for every possible source of data.

3 Specification of NADF File Format

A NADF file is a sequential file of records in NADF format. This format is flexible and allows a straightforward implementation of format adaptors.
A NADF record consists of:

- a four bytes integer representing the length (in bytes) of the whole NADF record (including the length field);
- a certain number of contiguous audit data fields. Each audit data field contains the three following contiguous items:

  identifier: an unsigned short (16 bit) integer which is the identifier of the audit data. This item must be aligned on a 2-bytes boundaries;

  length: an unsigned short integer which is the length of the audit data value;

  value: the audit data value itself.

In addition, audit data identifiers appearing in a NADF record must be sorted in a strict ascending order. This is important for ASAX to preprocess efficiently audit records before analysis. Figure 1 shows the general layout of a NADF record.

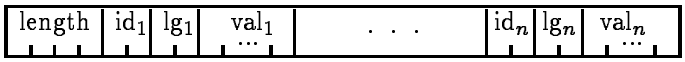

Figure 1: General NADF record layout

Audit Data Alignment
It follows from the alignment restriction on audit data identifiers that if the audit data value does not have an even number of bytes, a padding byte (a space ␣) must be appended to the value. This byte is not taken into account in the length of the value.

NADF record alignment
Since most machine architectures require 4-bytes integers to be 4-bytes aligned, a NADF record must always begin at a 4-bytes aligned area. This is especially important if NADF records are buffered for I/O. Similarly, if the total number of bytes in a NADF record is not a multiple of 4, padding bytes (spaces) must be added at the end of the record. Again, the record length does not include the padding bytes (see Example below).

Finally, a NADF file always begins with a header record having the following structure:

```
struct {
     int len;
     char val[12];
} TypeNADF = {15,"__NADF__1|\0"};
```

immediately followed by a padding space character ␣.
I/O routines on NADF files (see nadfio(3ASAX)) always check for the existence of the header record before going any further. This avoids processing non-NADF files.

Note that audit data identifiers are assigned arbitrarily, provided they are all distinct. Each NADF record may contain a different number of audit data.

Example
Suppose we want to convert to NADF format a record having the C declaration:

```
struct {
     char directory[10];
     int uid;
     char filename[12];
} Record = {"/tmp", 123, "/etc/passwd"};
```

and suppose the fields *directory*, *uid* and *filename* are assigned the identifiers 4, 1 and 2 respectively. The corresponding NADF record is depicted in Figure 2.

In particular, notice that strings in the NADF record are *not* null-terminated.

4 Guidelines for implementing Format Adaptors

In implementing a Format Adaptor for your native file format, we suggest following the two steps here under:

Figure 2: NADF record

Audit Data Description File

- examine carefully the documentation of the software producing the native files to figure out the exact structure of your native records;
- for each field in the data structure of your native record, assign a unique identifier number (between 0 and 65536). If a data field appears in many records, it is a good idea to assign them the same identifier;
- choose an external name for each field in the native record. This name is often the same as in the original record. Field names are referenced by this external name in the RUSSEL language;
- mapping between external field names and the identifiers must be written in what we call *audit data description file*. This is a simple text file where each audit data is described by a sequence of 5 lines. For easier parsing, the first line begins with '1' followed by a blank, the second line begins with '2' followed by a blank and so on. The first line in each sequence indicates the audit data identifier, the second indicates the type of the audit data in the native record. The third one indicates the type of the audit data in the NADF record. The second and third line are not interpreted for the present ASAX version but are useful for your own documentation. The fourth line indicates the external name and finally, the fifth line contains a free comment about the meaning of the audit data and is not interpreted by ASAX. At the beginning of the file you can optionally write additional comments. For instance, these lines may contain the version number, machine type, operating system type, etc. These lines must appear as follows: 0 or more lines beginning with an 'A' followed by 0 or more lines beginning with a 'B', and so on up to 'F'. A sample data description file is shown in the appendix

See ASAX user guide for the BNF syntax of audit data description files.

Format Adaptor implementation

Here is a skeleton of a Format Adaptor program:

```
Begin
   open native file
   create a NADF file /* using {\em creat\_NADF}() */
   allocate buffer for input (native) record
   allocate buffer for output (NADF) record
   read the first native record
   While not end of native file do
      begin
          convert native record to NADF format in the output buffer
```

```
            /*
               this is done by converting each field to a sequence of
               identifier, length and value}. Remember that identifiers
               must be sorted in a strict ascending order. You should
               convert fields with lower identifiers first.
            */
            write it to NADF file /* using {\em write\_NADF}() */
            read the next native record
         end
      close native file
      close NADF file /* using {\em close\_NADF}() */
      release input and output buffers
End.
```

Although you can use common I/O system calls (create, open, write, close), it is highly recommended that the standard NADF I/O routines (see nadfio(3ASAX)) be used instead. These routines support NADF file I/O at the record level and insert padding bytes between adjacent records to ensure proper data alignment. Also, *creat_NADF*() writes systematically the header record when it creates a new NADF file.

5 Final Remarks

From our experience in developing format adaptors, some misunderstandings often occur, so we provide further warnings:

- a NADF file is not an ASCII file but it is in a binary format. If you produce your NADF file on a given machine architecture and analyze it on a different architecture, some byte and bit ordering problems may arise. (For instance, MC68010 and MC68020 are big-ending with respect to bytes but little-endian with respect to bits. By contrast, the VAX and Intel 80386 are both little-endian for bytes and bits);
- audit data identifiers must be sorted in ascending identifiers. This is crucial for record preprocessing by ASAX during analysis;
- in converting a string data field to the format *identifier, length* and *value*, the null-character \0 marking the end of the string (like in C) is useless since we already know the string length (see Example before).

A A Sample Audit Data Description File

```
A Audit Data Description File for SunOS 4.1.1
B sun4c
C 1993-04-15 10:10:23
D ASAX V1.0

1 5
2 long
3 long
4 au_time
5 time since epoch

1 6
2 int
3 int
4 record_type
5 record type os the audit record

1 7
2 int
3 int
4 uid
5 user id

1 8
2 int
3 int
4 pid
5 process id

1 9
2 string
3 string
4 filename
5 the file name
```