NCSA Technical Report

HDF5 Packet Table API

A Joint Project between Boeing and University of Illinois

February 2005

Mike Folk, Quincey Koziol, James Laird, Elena Pourmal, NCSA, Univ. of Illinois
John Wegener, George Lewandowski, Joe Turner, Rodney Davis, Boeing Flight Test Instrumentation

Introduction

The proliferation of sensors and other instruments introduces enormous challenges to data management. Even for a single event, incoming synchronized time-sequenced data can have many sources, and the number of incoming data streams, as well as the types of data, can be large. In Boeing's flight test data applications, for instance, data arrives from test aircraft, voice communications, video, ground, satellite tracking, and other sources. This data must be gathered, integrated, processed, visualized, and archived. Similar scenarios exist for many different applications, such as environmental monitoring, vehicle testing, and medicine.

The collection and storing of these kinds of data historically have been reduced to unique in-house implementations. There is surprisingly little sharing of these infrastructure technologies even within an application area, let alone across application domains, resulting in frequent and costly re-invention of the same technologies.

HDF5 provides, in a single package, many of the capabilities that otherwise have to be developed from scratch. HDF5 can store virtually any kind of scientific or engineering data and to mix any number of objects of different types in a single container. HDF5 can support different access patterns, simplified data integration, datatype translation, fast I/O, and visualization and analysis software.

Uses of HDF5 Packet Table

The following examples describe typical uses of the HDF5 Packet Table API.

Writing homogeneous, fixed-sized packets of instrument data. An experimenter wants to store measurements from a fixed number of sensors, All measurements taken at the same time are collected and transmitted in a "packet", including a time stamp. The packet contents are to be saved as a record in a table. (Figure 1.) To do this, the experimenter can create an HDF5 datatype representing a fixed-size "packet" of data in an HDF5 file. This can be any atomic HDF5 datatype, or it could be a compound datatype. Each packet corresponds to a fixed-length horizontal entry, or "record," in a table.

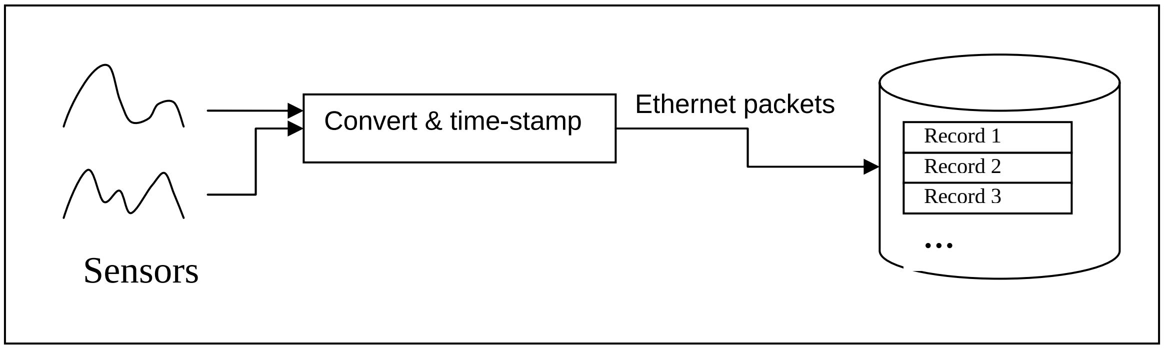

Figure 1. Fixed length packet example.

Writing data from multiple sources to a variable-length packet table. An experimenter wishes to store, in a single table, data from a number of sensor sources. All of the entries in a given packet have the same time stamp, but individual packets can vary in size and content. (Figure 2.) To do this, the experimenter can create an HDF5 datatype representing a variable-sized packet of data in an HDF5 file. Each packet corresponds to variable-length record in a table.

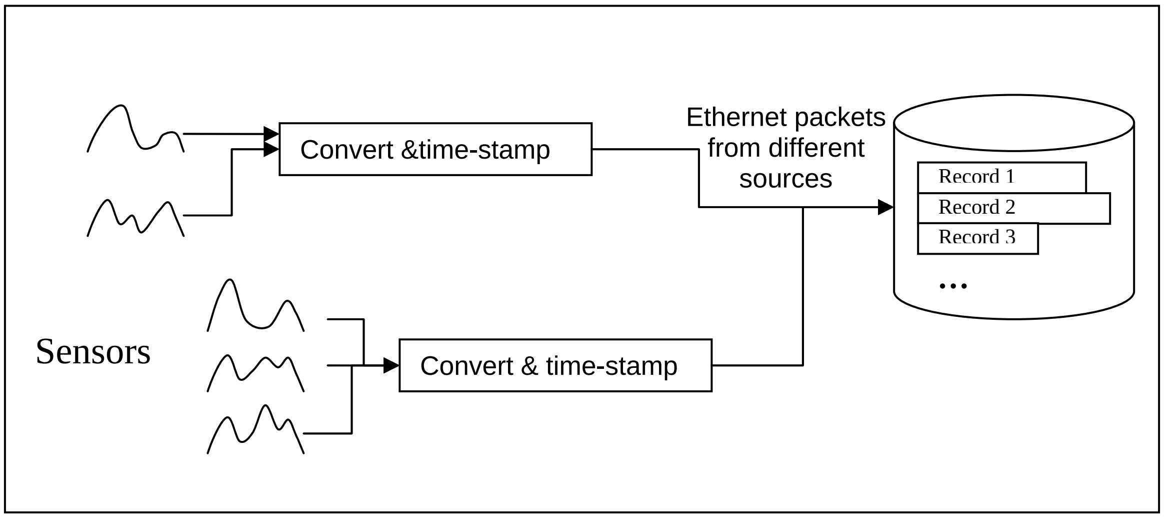

Figure 2. Variable length packet example.

HDF5 Packet library for flight test data

Boeing's Flight Test Instrumentations Group and the HDF5 development group at the University of Illinois have developed a library that is particularly suited for "packet" data, data that arrives in streams of packets from instruments at potentially very high speeds.

Fixed and variable length packets. A key to a general-purpose packet I/O library is that, at a certain level, all packet data falls into two simple structural categories: fixed-length packets and variable-length packets (Figure 3). In the fixed-length scenario, every record to be stored is of the same length. In the variable-length scenario, the data length will be variable for each storage time. The length may vary quite widely (even wildly) from one time to another.

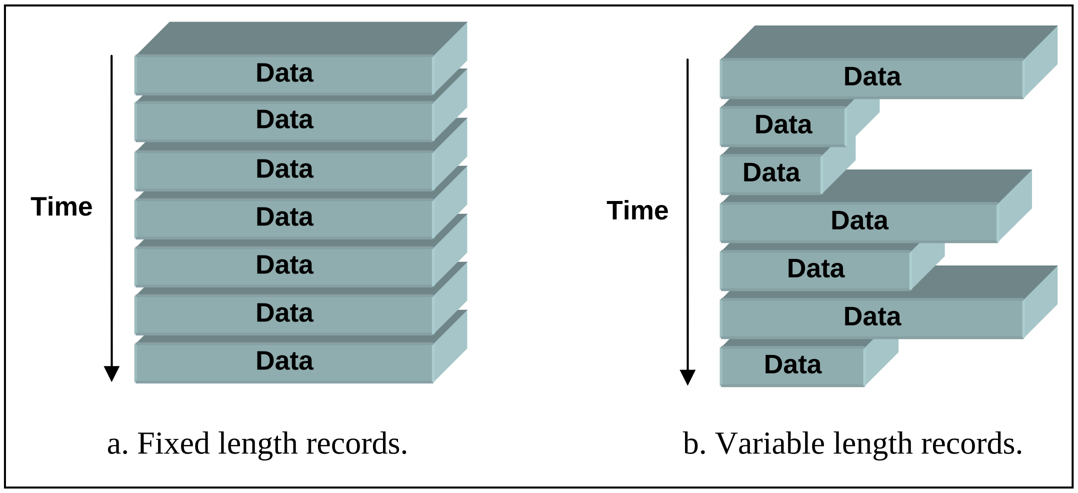

Figure 3. Fixed length and variable length packets.

To obtain efficient storage and I/O in HDF5, fixed length packets are stored differently in HDF5 files from variable length packets. The HDF5 packet library can create Packet Tables containing either fixed length or variable length records and provides similar APIs to access both.

The Packet Table implementation includes a C API, as well as prototype C++ wrapper objects, which are described in the Packet Table Reference Manual.

HDF5 "Table" vs. "Packet" APIs. The HDF5 packet API is an extension of the HDF5 "Table" API, whose prefix is H5TB. The H5TB Table is designed to be as high-level as possible, adding attributes and abstracting away much of HDF5's internal workings. In contrast, the Packet Table API is a lower-level approach, designed to maximize throughput (as is needed in real-time applications). The Packet Table thus provides bare-bones table functionality with essentially no overhead.[1]

Programming with the Packet library

Packet Tables have states, and must be opened and closed like HDF5 objects. Thus the Packet Table programming model is similar to that for working with other types of HDF5 objects:

1. An HDF5 file is created or opened.
2. The location of the object of interest is obtained.
3. The Packet Table is created or opened using the Packet Table API.
4. Operations are carried out on the Packet Table.
5. The Packet Table is closed
6. Other relevant objects are closed as necessary.

The Packet Table API covers steps 3-5. See the HDF5 documentation for information on how to do the other steps.

[1] Since the Packet Table API is currently under development, it is not yet included in the H5HL library. Its API may still be subject to change.

The following operations can be performed on Packet Tables:

1. Create and open Packet Table
   a. Create packet table with fixed-size record
   b. Create packet table with variable-length record
2. Close Packet Table
3. Write and read Packet Table
   a. Append packet(s)
   b. Position pointer for read operation
   c. Read packet(s) starting at specified position in the table
4. Query properties of Packet Table
   a. Number of packets in the Packet Table
   b. If an HDF5 object is a Packet Table
   c. If a Packet Table is of a variable-size type
5. Memory management for variable-size Packet Table

Packet Table APIs implement those operations. Full descriptions of Packet Table APIs can be found at http://hdf.ncsa.uiuc.edu/RFC/HDF5Packet/PacketTableRM.htm

Examples

These two examples demonstrate the use of the C language version of the Packet Table API

1. Fixed length example. Create packet table with fixed-length records and perform writes and reads.

```
#include "H5TB.h"
#include <stdlib.h>

/*-------------------------------------------------------------------------
 * Packet Table Fixed-Length Example
 *
 * Example program that creates a packet table and performs
 * writes and reads.
 *
 *-------------------------------------------------------------------------
 */

int main(void)
{
 hid_t          fid;        /* File identifier */
 hid_t          ptable;     /* Packet table identifier */

 herr_t         err;        /* Function return status */
 hsize_t        count;      /* Number of records in the table */

 int            x;          /* Loop variable */

    /* Buffers to hold data */
 int writeBuffer[5];
 int readBuffer[5];

   /* Initialize buffers */
 for(x=0; x<5; x++)
 {
     writeBuffer[x]=x;
     readBuffer[x] = -1;
 }

    /* Create a file using default properties */

fid=H5Fcreate("packet_table_FLexample.h5",H5F_ACC_TRUNC,H5P_DEFAULT,H5P_DEFAU
LT);

    /* Create a fixed-length packet table within the file */
    /* This table's "packets" will be simple integers. */
 ptable = H5TBmake_packet_table(fid, "Packet Test Dataset", H5T_INTEGER, 1);
 if(ptable == H5I_INVALID_HID)
     goto out;

    /* Write one packet to the packet table */
 err = H5TBappend_single_packet(ptable, &(writeBuffer[0]) );
 if(err < 0)
     goto out;

    /* Write several packets to the packet table */
 err = H5TBappend_packets(ptable, 4, &(writeBuffer[1]) );
 if(err < 0)
     goto out;

    /* Get the number of packets in the packet table.  This should be five.
*/
 err = H5TBget_num_packets(ptable, &count);
 if(err < 0)
     goto out;

 printf("Number of packets in packet table after five appends: %d\n", count);

    /* Initialize packet table's "current record" */
 err = H5TBcreate_ptable_index(ptable);
 if(err < 0)
     goto out;

    /* Iterate through packets, read each one back */
 for(x=0; x<5; x++)
 {
    err = H5TBget_next_packets(ptable, 1, &(readBuffer[x]) );
    if(err < 0)
      goto out;

    printf("Packet %d's value is %d\n", x, readBuffer[x]);
 }

    /* Close the packet table */
 err = H5TBclose_packet_table(ptable);
 if(err < 0)
     goto out;

    /* Close the file */
 H5Fclose(fid);

 return 0;

 out: /* An error has occurred.  Clean up and exit. */
    H5TBclose_packet_table(ptable);
    H5Fclose(fid);
    return -1;
}
```

2. Variable length example. Create packet table with variable-length records and perform writes and reads.

```
#include "H5TB.h"
#include <stdlib.h>

/*-------------------------------------------------------------------------
 * Packet Table Variable-Length Example
 *
 * Example program that creates a variable-length packet table and performs
 * writes and reads.
 *
 *-------------------------------------------------------------------------
 */

int main(void)
{
 hid_t          fid;        /* File identifier */
 hid_t          ptable;     /* Packet table identifier */

 herr_t         err;        /* Function return status */
 hsize_t        count;      /* Number of records in the table */
 int            x;          /* Loop variable */

    /* Buffers to hold data */
 hvl_t writeBuffer[5];
 hvl_t readBuffer[5];

    /* This example has two different sizes of "record": longs and shorts */
 long longBuffer[5];
 short shortBuffer[5];

   /* Initialize buffers */
 for(x=0; x<5; x++)
 {
     longBuffer[x] = -x;
     shortBuffer[x] = x;
 }
```

```
    /* Fill the write buffer with a mix of longs and shorts */
    /* We need to supply the length of each record and a pointer to */
    /* the beginning of each record. */
 writeBuffer[0].len = sizeof(long);
 writeBuffer[0].p = &(longBuffer[0]);
 writeBuffer[1].len = sizeof(short);
 writeBuffer[1].p = &(shortBuffer[1]);
 writeBuffer[2].len = sizeof(long);
 writeBuffer[2].p = &(longBuffer[2]);
 writeBuffer[3].len = sizeof(long);
 writeBuffer[3].p = &(longBuffer[3]);
 writeBuffer[4].len = sizeof(short);
 writeBuffer[4].p = &(shortBuffer[4]);

    /* Create a file using default properties */

fid=H5Fcreate("packet_table_VLexample.h5",H5F_ACC_TRUNC,H5P_DEFAULT,H5P_DEFAU
LT);

    /* Create a variable-length packet table within the file */
 ptable = H5TBmake_vl_packet_table(fid, "Packet Test Dataset", 1);
 if(ptable == H5I_INVALID_HID)
     goto out;

    /* Write the entire buffer to the packet table */
 err = H5TBappend_packets(ptable, 5, writeBuffer );
 if(err < 0)
     goto out;

    /* Get the number of packets in the packet table.  This should be five.
*/
 err = H5TBget_num_packets(ptable, &count);
 if(err < 0)
     goto out;

 printf("Number of packets in packet table after five appends: %d\n", count);

    /* Read all five packets back */
 err = H5TBread_packets(ptable, 0, 5, readBuffer );
 if(err < 0)
    goto out;

 for(x=0; x<5; x++)
 {
    printf("Packet %d's length is %d\n", x, readBuffer[x].len);
    if(readBuffer[x].len == sizeof(long))
        printf("Packet %d's value is %d\n", x, *( (long *) readBuffer[x].p)
);
    else
        printf("Packet %d's value is %d\n", x, *( (short *) readBuffer[x].p)
);
 }

    /* Before we close the packet table, we must free the memory that */
    /* the pointers in readBuffer point to. */
 err = H5TBfree_vlen_readbuff(ptable, 5, readBuffer);
 if(err < 0)
     goto out;

    /* Close the packet table */
 err = H5TBclose_packet_table(ptable);
 if(err < 0)
     goto out;

    /* Close the file */
 H5Fclose(fid);

 return 0;

 out: /* An error has occurred.  Clean up and exit. */
    fprintf(stderr, "An error has occurred!\n");
    H5TBclose_packet_table(ptable);
    H5Fclose(fid);
    return -1;
}
```