FastArchive Library Reference

Ben Smith

Introduction

FastArchive is a C++ class that provides functionality for the storage of large numerical data sets in platform-independent compressed binary files. FastArchive also produces a C++ header file containing code for the rapid retrieval of data from the archive via an iterative search algorithm. Each data record to be stored must be identified by a unique numerical index value or combination of values.

Compilation and linkage

Linux/Unix: A make file (GNU g++ compiler) is included with the release. This builds FastArchive as a simple object file (`fastarchive.o`) which may be linked in to any C++ program; e.g.

```
$ g++ xampl.cpp fastarchive.o
```

Windows: Build FastArchive as a static library (`fastarchive.lib`). Include the library in the build for any C++ application. Alternatively, include `fastarchive.cpp` in the build for any C++ application.

All platforms: All of the following `#includes` must appear at the start of any source code file invoking functionality from FastArchive, i.e. any program *writing* data to an archive. Note that `fastarchive.h` must appear *last* in the list; the full pathname may be required:

```
#include <stdio.h>
#include <stdlib.h>
#include “fastarchive.h”
```

Programs that *read* data from an archive need not include `fastarchive.h`, but must instead include the customised header file generated when the archive was created. The name is based on the *name* argument to function **newarchive** (see below) with a ‘.h’ extension (e.g. if *name* was MyData, the header file will be called `MyName.h`). In addition, programs reading from an archive must include `<string.h>` before the customised header file; for example:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include “MyData.h”
```

Creating a fast archive

Applications creating a fast archive should link the FastArchive library. Declare an instance of class CFastArchive and declare a name for the archive. This name will appear in the

filename of the archive itself (suffixed by '.bin'), the customised header file providing code for extracting data from the archive (suffixed by '.h'), and the class providing this functionality (with an upper-case first letter and suffixed by 'Archive':

```
CFastArchive ark;
ark.newarchive("MyData");
```

Specify one or more index items by which records in the archive will be identified. The index items may be integer or floating point values and in combination they must identify each record uniquely (i.e. there must not be two or more records with the same values for all of the index items). A minimum and maximum value and a minimum precision is specified for each item; e.g.

```
ark.defineindex("longitude",-180.0,180.0,0.5);
ark.defineindex("latitude",-90.0,90.0,0.5);
      // Each record is identified by a unique lat/long coordinate
      // to the nearest half degree
```

Specify one or more numerical items to include in each record. Each item is an array of one or more values. Individual values should be specified as an array of dimension 1:

```
ark.defineitems("country_id",1,0,500,1);
ark.defineitems("monthly_temperature",12,-50.0,50.0,0.1);
```

Data may now be written to the archive. For a single record, the code may look something like this:

```
double this_longitude,this_latitude;
double temp[12];
double dval;
int this_country;

.
.
.

ark.setindex("longitude",this_longitude);
ark.setindex("latitude",this_latitude);
dval=double(this_country);
ark.setitem("country_id",&dval);
      // don't forget the '&' with non-array arguments
ark.setitem("monthly_temperature",temp);
ark.storerecord();
```

When all records have been written, the archive must be closed:

```
ark.closearchive();
```

If there are no errors, this will create a binary file (`MyData.bin`) containing the archive and a header file (`MyData.h`) defining class MyDataArchive with functionality for retrieving data from the archive and class MyData which can store data for a single record. Both the archive itself and the generated code for reading it should be portable across platforms.

Reading a fast archive

Applications reading data from a previously created fast archive do *not* need to link to the FastArchive library. Instead, they should #include the generated header file (see above). First, declare an instance of the archive class (MyDataArchive in the example above) and open the archive file:

```
MyDataArchive ark;
bool success=ark.open("MyData.bin");
if (!success) {
      printf("Could not open archive for input\n");
      exit(99);
}
```

Individual records may then be retrieved based on their index value(s):

```
MyData rec;
int m;
rec.longitude=this_longitude;
rec.latitude=this_latitude;
success=ark.getindex(rec);
if (success) {
      printf("Country code = %g\n",rec.country_id[0]);
      for (m=0;m<12;m++)
           printf("Month %d temperature = %g\n",
                      m,rec.monthly_temperature[m]);
}
else printf("Record at (%g,%g) not found\n",
      rec.longitude,rec.latitude);
```

To retrieve all records in the order they appear in the archive, use code similar to this:

```
bool flag;
flag=ark.rewind();
while (flag) {
      flag=ark.getnext(rec);
      if (flag) printf("Loaded record at (%g,%g)\n",
           rec.longitude,rec.latitude);
}
```

The archive is closed by:

```
ark.close();
```

Function synopsis

Public member functions of class CFastArchive

void **newarchive**(char* *name*)

Create a new archive file with specified name. A directory part may be included. The specified name should not include an extension but '.bin' will be added to the name of the archive file created, and '.h' to the name of the customised C++ header file for the archive.

void **closearchive**()
Close the archive file opened with newarchive(). This function *must* be called, otherwise the archive file will not be readable.

short **defineindex**(char* *name*, double *minval*, double *maxval*, double *precision*)
Add a new index item (i.e. variable) to the data structure of the archive. Values will (and must) be in the range *minval-maxval* and will be stored at the specified precision. An index item is used in the application program as part of the key to retrieve a specified record from the archive.

short **defineitems**(char* *name*, int *nitem*, double *minval*, double *maxval*, double *precision*)
Add a new item to the data structure of the archive. Values will (and must) be in the range *minval-maxval* and will be stored at the specified precision.

void **setindex**(char* *name*, double *val*)
Set the value for the specified index item in the current record. The index item must have been previously declared by a call to **defineindex**().

void **setitem**(char* *name*, double* *val*)
Set the value for the specified item in the current record. The item must have been previously declared by a call to **defineitems**().

void **storerecord**();
Store the current record in the archive file. The values of all the index item(s) should have been previously set with **setindex**(). The values of all the item(s) should have been previously set with **setitem**().

Public member functions of the generated archive class

These functions are available in the class (e.g. MyDataArchive) defined in the generated archive header file.

bool **open**(char* *filename*)
Attempts to open the specified file as a fast data archive. The archive and header file should have been produced together by the same program using class CFastArchive). Returns false if the file could not be opened or had format errors. **open**() with no argument will attempt to open the archive with its original file name.

void **close**()
Closes the archive (if open).

bool **rewind**()
Sets the file pointer to the first record in the archive file. Returns false if no archive file is currently open.

bool **getnext**(MyData& *obj*)
Retrieves the next record in the archive file and advances the file pointer to the next record. Data are written to the member variables of *obj*. Returnd c1.22(le)]TJT*0.0003 Tc-0.008Tc-0.00

bool **getindex**(MyData& *obj*)

Searches the archive for a record matching the values specified for the index items in *obj*. If a matching record is found, the data are written to the member variables of *obj*. Returns true if the archive was open and a matching record was found, otherwise false. The search is iterative and fast.

Contact information

Direct any queries to:

Dr Ben Smith
Dept of Physical Geography and Ecosystems Analysis
Lund University
Geocentrum II
22362 Lund
Sweden

E-mail: ben.smith@nateko.lu.se
URL: www.nateko.lu.se/personal/benjamin.smith

22 July 2006.