

ANN/Objectivity: User Manual.

Gabriele Garzoglio (garzogli@fnal.gov), May 2001.

Table of Contents

Table of Contents.....	1
Introduction.....	1
Where to get ANN/Objectivity	1
Setup	1
Installation	2
Running a Test.....	3
How-to for the normal operations.....	3
The tag extractor	4
The definition of the space and the schema.....	5
ANN/Objectivity and the Global File System	5
A sample API.....	6
The Corba interface	8
Trouble shooting.....	10
The original ANN library license	10
Appendix A: schema and space definition sample	10

Introduction

These are the instructions on how to use the ANN/Objectivity Nearest Neighbors engine. This engine has been built on top of the original ANN library (David M Mount, University of Maryland). Almost all of the options of the library are still available and are thoroughly described in: “The ANN Programming Manual”; David M. Mount, Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, www.cs.umd.edu/~mount/ANN/.

Where to get ANN/Objectivity

The current version of ANN/Objectivity as well as all the development versions can be found at the computing division cvs repository:

```
setenv CVSROOT \ cvsuser@cdcvs.fnal.gov:/cvs/cd
```

The module name is ANN.

Setup

Ann/Objectivity depends on these products:

Objectivity (v6_0_3 has been used)

Orbacus (v4_0_5 has been used).

ANN/Objectivity has been developed on Red Hat 7 with gcc 2.96.

Part of the development has been done on an SGI Origin 2000. On this platform, ANN has been tested up to the cvs symbolic tag name “MultiDBsgi”.

There is a script on the top directory called `ann.source` (the script uses Fermilab UPS). Sourcing it from a csh shell as

```
source ann.source
```

the correct environment to build and run ANN is set.

Installation

-First of all source `ann.source`. This sets up products using Fermilab UPS.

-To install a test version of ANN/objectivity:

1. make sure the objectivity lock server is running (type as root: `>ools &`)
2. edit `objy/Makefile`: write you objectivity lock server in `LS_HOST`; you may need to change `FDID` to avoid locking conflicts with other federations with the same number.

3. from the top dir type

```
>make linux_test
```

This compiles ANN/objectivity building the test database `objy/DB/ANNdb`.

WARNING: this uses the test schema `objy/ANNdbDDL.ddl.old_tag`, based on the "old" (before Feb 2001) `sx` tag. This old schema overwrites `objy/ANNdbDDL.ddl`, which is the schema used at compilation time: remember to replace it with a non-test schema (i.e. `cp objy/ANNdbDDL.ddl.new_tag objy/ANNdbDDL.ddl` when using the tag extractor for the new `sx` tag)

4. to clean up the built code AND the test database

```
>make realclean "BOOT=DB/ANNdb"
```

To clean up just the built code

```
>make clean
```

-To install a version of ANN/objectivity that works with the new (2001) `sx` tag.

1. make sure the objectivity lock server is running (type as root: `>ools &`)
2. edit `objy/Makefile`:
 - *write your objectivity lock server in `LS_HOST`; you may need to change `FDID` to avoid locking conflicts with other federations with the same number.

*write the boot file of your tag database in `BOOT`. On the `gfs` area of TAM there is a tag database in `/gfs/sci/gg/EDRTAG/SXDB` and a few smaller test databases is `/gfs/sci/gg/DBTAG?/SXDB`. You can use `objy/ANNdbDDL.ddl.new_tag` as schema for these databses.

If you want to create a tag database using the tag extractor, use the `objy/ooRunTagExtr.csh` utility (help typing `objy/ooRunTagExtr.csh`): this builds the tag extractor and extracts the tag database. Note that you may need to modify this script if you plan to use it on a different `SX` database than the EDR version.

3. from the top dir

```
make linux
```

This builds `ann/objectivity` with your own schema.

WARNING: check that `objy/ANNdbDDL.ddl` is the schema you want to use (i.e. `cp objy/ANNdbDDL.ddl.new_tag objy/ANNdbDDL.ddl` when using the tag extractor for the new `sx` tag). Note that `>make linux_test` overwrites this schema with the test schema.

4. to clean up the built code AND the current database

```
>make realclean
```

To clean up just the built code

```
>make clean
```

Running a Test

The samples/ann_sample program can be used to re-cluster the database and search through it. See the paragraph “*A sample API*” to a complete description.

For now you can use it to run a test typing

```
>cd samples
>./ann_sample -fd ../objy/DB/ANNdb -db TAGS -b 10 \
-dfo ../objy/DB/tree.dump -r 200 -s std -nn 5 > nn.out
```

This command operates on the test federation ANNdb; the database “base name” is TAGS (in fact the databases are called TAGS_1, TAGS_2...; this naming convention is used by the tag extractor also); it builds a tree with a bucket size of 10 objects; it dumps a text version of the tree calling the file tree.dump; it re-clusters the database writing 200 object per new database (this low number is good for testing only) and saves the new tree as tree.dump__ANNrecl; searches through the whole dataset using the standard (std) search algorithm and gives in output 5 nearest neighbors per point.

The output file includes log directives, Objectivity status information and the actual nearest neighbors. This API prints out the object index within the database, instead of the whole object (see the code for examples of direct object print out). The distance is the Euclidean distance to the query point (other distance metrics are configurable: see the “ANN programming Manual”). The status information and the file ann_perf.dat have been used to benchmark ANN, and can be avoided specifying it at compilation time (see “*A sample API*”).

How-to for the normal operations

When I speak about “normal operations” I’m considering a scenario where there is a stable large tag database (updated once every 6 months) (see “*The Tag extractor*” and “*ANN/Objectivity and the Global File System*”) and analysis to be implemented using the k^{th} nearest neighbors engine. I expect a relatively long phase of development of the analysis program, which will make use of the ANN/Objectivity corba interface. There are two examples in corba/client/ of C implementations that use the interface; other attempts to wrap C programs that use the interface within TCL are under way (even if there are ORB implementations for TCL, they work only with very recent versions of the language i.e. 7.5 or above; upgrading already stable applications is not always straight forward). The test of the client is dependent on the next steps: the preparation of the ANN server.

The definition of the space where to look for nearest neighbors is a relevant issue for the analysis and concerns the ANN/Objectivity server, which is responsible to build and manage the tree: this is the main data structure used to perform the nn search. There will certainly be some trial and error iterations on this side: the paragraph “*The definition of the space and the schema*” gives a “real life” example of how to define such a space. The user will modify ANNdbDDL.ddl, using as a template (ANNdbDDL.ddl.new_tag) and changing essentially only the number of dimensions of the space and the method fetch(int i).

Once ready, the objy/Makefile has to be tailored: the tag BOOT for the boot file for the federation currently in use, the tag LS_HOST for the lock server host used (any of the tam machines). In the objy/ directory, `> make linux` compiles the .ddl file i.e. the user defined ANN space; checks the schema against the federation in use (there should be no change), compiles and install the libraries that ANN uses to access the Objectivity database (access.C) (again, see “*The definition of the space and the schema*”).

At this point `> make linux` in the src/ and corba/server/ directory should work fine. Some compilation options may be set to define, for example, special metrics of the space or the kind of tree to build (kd or bd) or to output benchmark information. The paragraphs “*The corba interface*”, “*A sample API*” and the “ANN programming Manual” give details on what can be done. Note that `> make linux` in the top directory automatically compiles everything. The same paragraphs describe how to run the server to build/save the tree and re-cluster the database. Once the server is run, the client can be tested.

Finally I provide a few *Trouble Shooting* hints, referring to some misleading situation.

The tag extractor

The tag extractor is a utility initially implemented by Koen Holtman (Caltech) to extract the PhotoTag objects from the SX database. The tags are written into a new federation, which ANN/Objectivity can interface to. The internal references to other persistent objects are not rewritten: the tags are made “flat”.

The source code of the Tag Extractor is in objy/TagExtractor/, the main program is dbScanTags.cpp. Some objectivity management tools must be used together with the TagExtractor to create the new federation of tags (see objy/TagExtractor/README). To make things as simple as possible, the procedure has been scripted in objy/ooRunTagExtr.csh. This script builds and runs the tag extractor.

WARNING: as it is, the script IS NOT general, since it deals with specific problems (database corruption) of the EDR version of the SX database. Its generalization should be straightforward.

To run the script type:

```
>ooRunTagExtr.csh new_fd_path old_fd_path new_fdnum
```

where

- old_fd_path is the path to the objectivity catalog from which the data are extracted,
- new_fd_path is the path to the new catalog,
- new_fdnum is the federation number of the new federation (must be different for every new federation)

Once the new federation is created, you can:

- take the schema definition of the new tag,
- add the information needed by ANN to identify the database with an n-dimensional space (see paragraph “*The definition of the space and the schema*”),
- name this file objy/ANNdbDDL.ddl,
- include the new federation path and federation number in the objy/Makefile
- run `make linux` from the top directory.

This should link ANN/objectivity with the new federation.

NOTE: while you may not need to change the file objy/ANNdbDDL.ddl, at least in the schema part, because you are use the standard PhotoTag schema, YOU WILL NEED TO REBUILD ANN TO LINK IT WITH THE NEW FEDERATION.

Note that some improvement on the search speed of ANN could be obtained by tuning the number of containers per database and their growth factor at extraction time. This part is not implemented, even if the clustering directives are specified on an object-by-object basis. Please, refer to “Using Objectivity/C++” manual, chapter 21 “Monitoring and Tuning Performance”, in particular “Tuning for Runtime Speed” paragraph, for details.

The definition of the space and the schema

To interface with the database and build the tree, ANN/Objectivity needs to know

1. the definition of the tag object stored in the database (schema);
2. the number of dimensions;
3. the definition of the space on which it has to build the tree (e.g. ra/normalization, dec/norm., red_shift/norm, colors).

This information is provided in the file objy/ANNdbDDL.ddl. This file is divided in 3 sections corresponding to the above points. Appendix 1 shows an example used in a real analysis: here the space is 5-dimensional, the schema is the old (prior to Feb 2001) SX tag object, and the space is a normalization of the red shift, position and color parameters. The normalization function is provided via a user defined lookup table (only partially reported here).

First of all the number of dimensions have to be specified, changing `#define FIXED_DIMENSIONS 5` and the definition of the `class ANNpointObj`: the generalization is easy following the template.

The original schema of the objectivity tag database can be cut and pasted. Note that this schema must not have any references to other persistent objects: the tag must be flat. Modify the database and the schema accordingly if this is not the case.

The definition of the space is specified within the method `ANNpointObjPers::fetch(int i)`. In fact, ANN represents the space as a vector of `ANNpointPers` objects, each of which is a wrapper for an objectivity reference to a persistent object of class `ANNpointObjPers`. The method `ANNpointPers::operator[]`, used by all the methods of ANN to access the i^{th} dimension of the space, simply returns what `fetch(i)` returns for i . Appendix A shows a clear example of the definition a 5 dimensional space.

ANN/Objectivity and the Global File System

As mentioned on the “ANN performance report”ⁱ (May 2001), Objectivity doesn’t see as shared disks mounted via the Global File System (see the “Objectivity /DB Administration” manual, Chapter 2 “Specifying Remote and Local Files” for more details on the recognized file systems).

To run the same database on multiple machine of the TAM cluster, it is possible to overcome the problem

- copying the federation catalog to a new directory,
- creating links to the physical databases in the new directory,
- running the `ooinstallfd` utility from the new machine.

This procedure is scripted in oby/ooLinkAnnTagDb.csh (run the script with no parameters for help)

A sample API

The C++ program samples/ann_sample.cc is a fairly complete interface to all the feature of the ANN/Objectivity library. Here is a description of its arguments. From the help:

Usage:

```
./ann_sample [-d dim] [-nn k] [-e eps] [-b bckt_size] \  
[-s pri|std|no] [-dfi dump_in] [-dfo dump_out]\  
[-nfiles n_files] [-npages init_pages] [-nmaxpages max_pages]\  
[-r num_tag] -fd fd -db db
```

where:

dim	dimension of the space (default FIXED_DIMENSION)
k	number of nearest neighbors per query (default 1)
eps	the error bound (default 0.0)
bckt_size	# of points in the leaf nodes (default 10)
pri std no	search algo: priority ,standard (default), no search
dump_in	name of an old tree dump file. Do NOT build a new tree
dump_out	name for tree dump file.
n_files	num of objy db files open together when ooinit() (def. 12)
init_pages	objy initial num. of pages when ooinit() (def. 200)
max_pages	objy max num. of pages when ooinit() (def. 500)
num_tag	reclusterize built tree writing num_tag tags per db.
fd	path to the objectivity federation boot file
db	base name of the database

Results are sent to the standard output.

If not specified, the dimension of the space is the one set in the file oby/ANNdbDDL.ddl as #define FIXED_DIMENSION. You cannot run an analysis on a number of dimensions greater than FIXED_DIMENSION, but you can run an analysis in a subspace; in other words, dim can be less or equal to FIXED_DIMENSION.

The bucket size is the maximum number of points stored in a leaf node of the tree. As discussed in the “ANN performance report”, the bucket size must be chosen as a trade off between memory availability and search speed. If the bucket size is too big, the tree is short (many point stored together in the same leaf node), but the search is slow (in the

leaf node the search is the standard $O(N^2)$). For database with less than 10 million object, a bucket size of 10 seems a reasonable trade off.

If the option `-dfo` is specified, ANN builds a tree and dumps a text representation of it with the file name specified. Note that if the option `-r` is specified (re-cluster the database), also a representation of the new “re-clustered” tree is dumped, adding to the same name the suffix “`__ANNrecl`”. Once these files have been created, they can be reloaded to run analysis on the same tree running `ann_sample` with the option `-dfi`. WARNING: there is no check on the internal consistency between the database you specify with the options `-df`, `-db` and the dumped tree you specify with `-dfi`. In other words, do not specify a tree for a re-clustered database while using the non re-clustered one or vice versa.

The options `-nfiles`, `-npages`, `-nmaxpages` configure the objectivity client page size. Refer to the “Using Objectivity/C++” manual for a detailed explanation. The default should work fine for ANN/Objectivity in practically all the conditions.

The option `-r` creates new databases on the same federation, clustering the objects according to the index built when the tree is built. The maximum number of objects per database must be specified after `-r`. Using the new PhotoTag object, a good choice is 5,000,000 , which keep the size of the databases around 1.3 Gigabytes.

Note that if you want to delete a re-clustered database, you can simply run `objy/ooDelAnnReclDb.csh` .

The original ANN library offers a wide range of algorithms to build kd and bd-trees: in particular several possible splitting and shrinking rules are available; several different norms (defining the concept of distance within the parameter space) and other configuration options can be selected commenting or un-commenting `#defines` in the `include/ANN/ANN.h` file. Refer to “The ANN Programming Manual” for a very nice description of all these options.

You can decide whether making ANN/objectivity build a kd or a bd tree at compilation time, changing the header of `ann_sample.cc`: simply uncomment `KD_TREE` or `DB_TREE` (`BF_TREE` is a brute force search of the tree, there for debugging purposes). Here you can also decide if you want ANN to print out benchmarking information in the file `ann_perf.dat` and Objectivity status information on standard output: comment out `#define ANN_BENCHMARK` if you do NOT want them.

To represent graphically a 2 dimensional section of the tree, the program `tools/ann2fig` has been modified to work with Objectivity. See “*The ANN programming Manual*” for a detailed description. In addition to the original options, this version requires the name of the federation (`-fd`) and the database base name (`-db`). The following is a good example for the test dataset:

```
> cd samples
> ./ann_sample -s no -b 1 -fd ../objy/DB/ANNdb -db TAGS
  -dfo ../tools/dump_kd.ann > ! nn.out
> cd ../tools/
> ./ann2fig -fd ../objy/DB/ANNdb -sl 2.0 0.5 3.0 0.5 -db TAGS dump_kd
> xfig dump_kd.fig
```

The Corba interface

In order to leave maximum freedom in the choice of the programming language for the analysis program, it is available a version of ANN/Objectivity, which implements corba interfacesⁱⁱ. The analysis program has to be linked to the ANN/Objectivity interface and an ORB implementation for the chosen programming language: the analysis program acts as a client. The ANN/Objectivity server makes available all the functionalities that the sample API `ann_sample` provides (see “*A sample API*”): the server is run on a federation with the option of building/loading a tree and possibly re-clustering the database, and then the server loop begins.

The server implements 3 methods that the client can use: `init`, `search`, `searchNextIdx`. From the idl interface definition:

```
// Exceptions
exception ANNinitFail {
    string reason;
};
exception ANNsearchError {
    string reason;
};

// Type definition
enum Corbasearch_type {PRI, STD};
typedef sequence<double> ANNCorbapoint;
typedef sequence<ANNCorbapoint> ANNCorbapointArray;
typedef sequence<double> distCorbaArray;

// Interface
interface ANNCorbaSearch
{
    // Communicate to the server once for all the configuration
    // parameters. This method can be called again to change the server
    // configuration: it is part of the interface to reduce parameter
    // passing during analysis.
    void init(in short dim, in short k, in float eps,
             in Corbasearch_type searchType)
        raises(ANNinitFail);

    // Retrieve nn to the queryPt point in the database.
    void search(in ANNCorbapoint queryPt, out ANNCorbapointArray nn,
```

```

out distCorbaArray nnDist)

raises(ANNsearchError);

// Retrieve nn to the nextIdx point in the database.
// Note: nextIdx >= 0
boolean searchNextIdx(in long nextIdx, out ANNCorbapoint queryPt,
out ANNCorbapointArray nn,
out distCorbaArray nnDist)

raises(ANNsearchError);
};

```

An example of the implementation in C of a client is available in corba/client.

Note that by convention, the memory for the input parameters must be allocated and de-allocated by the client, while the memory for the output parameters is allocated by the server and must be de-allocated by the client. Subtle errors may arise if this convention is not followed.

The init method is used to configure the server, setting the parameters that are likely to stay constant for the entire duration of a nn search. This method is part of the interface to reduce the overhead due to parameter passing during the call of methods search and searchNextIdx, which presumably are called million times during an analysis. The input parameters are the number of dimensions, which must be \leq FIXED_DIMENSIONS as set in objy/ANNdbDDL.ddl (see paragraph “*The definition of the space and the schema*”), the number k of nearest neighbors, the error bound eps (0 exact search, > 0 for approximate: see “The ANN programming manual”), the search type: PRI for a priority queue search, STD for a standard search.

Note: the current server implementation keeps the parameters set by init in a single area of memory (single instantiation of the interface implementation class): this prevents clients with different configuration requirements to run concurrently. No run time check of the consistency of a configuration is made: it is the responsibility of the user to run clients concurrently only if they require the same configuration.

The method search accepts in input a query point as a dim-array, and fills in the dim*k-array of nearest neighbors and the k-array of distances to the query point. As in the case of ann_sample, the query point may be any point i.e. not necessarily part of the sample of points in the database.

The method searchIdx accepts in input a zero-based index and fills in the query point corresponding to that index in the database, the array of nn and the array of distances. The function returns FALSE if the index is out of range: this way the client can run over the entire database implementing a loop for which the test is the call to the method searchIdx.

As for ann_sample, the compilation of the server can be configured to printout benchmarking information and whether to build a kd or a bd-tree by commenting/uncommenting the #definition in the file corba/server/ANNCorba_config.h

Note that at this stage of development, the client needs to find the file ANNCorba.ref in the directory ../. This file holds the location of the server and it is written by the server itself in the directory ../ when launched. This logistic works best if the server is launched from the directory corba/server/ and the client from corba/client/.

Trouble shooting

Most of the problems that I've encountered are related to Objectivity: for example, many times, after deleting a federation, I had to wait up to about 1 minute before being able to build a new federation; in my opinion this is due to the slow communication between application and lock server. At the beginning I've found this error misleading. Patience was the best solution.

The original ANN library license

Copyright (c) 1997-1998 University of Maryland and Sunil Arya and David Mount
All Rights Reserved.

This software and related documentation is part of the Approximate Nearest Neighbor Library (ANN).

Permission to use, copy, and distribute this software and its documentation is hereby granted free of charge, provided that

- (1) it is not a component of a commercial product, and
- (2) this notice appears in all copies of the software and related documentation.

ANN may be distributed by any means, provided that the original files remain intact, and no charge is made other than for reasonable distribution costs.

Appendix A: schema and space definition sample

This file has been used on March 2001 to look for cluster of galaxies in the southern sky catalog of the Sloan Digital Sky Survey.

```
File objy/ANNdbDDL.ddl
// THIS FILE MUST BE MODIFY TO OPERATE WITH A NEW SCHEMA AND SPACE
// DEFINITION.
// IT IS DIVIDED IN 3 AREAS:
// 1) DEFINITION OF THE NUMBER OF DIMENSIONS: change FIXED_DIMENSIONS
//     and add/remove dimensions from the ANNpointObj object.
// 2) ORIGINAL SCHEMA: in principle you should just make sure that
//     this tag has no references to other persistent object (i.e. it's
//     flat).
// 3) DEFINITION OF THE SPACE: the combination of object parameters
//     that identify the space. The tree is built (and the data indexed)
//     according to this.

#include <stdlib.h>
```

```

#include <fstream.h>                // file I/O

typedef    double    ANNcoord;    // coordinate data type

//DEFINITION OF THE NUMBER OF DIMENSIONS...
#define FIXED_DIMENSIONS 5

struct ArrayOutOfBound {
    ArrayOutOfBound(int dim) {
        cerr << "ANN error: expected " << FIXED_DIMENSIONS
            << " dimensions instead of " << dim+1 << ".\n";
    }
};

class ANNpointObj {    // a representation of a point
public:
    ANNcoord x0;
    ANNcoord x1;
    ANNcoord x2;
    ANNcoord x3;
    ANNcoord x4;

    ANNpointObj()
    { }

    ANNpointObj(const ANNpointObj &p) {
        x0 = p.x0;
        x1 = p.x1;
        x2 = p.x2;
        x3 = p.x3;
        x4 = p.x4;
    }

    ANNcoord& operator[](int i) {
        switch (i) {
            case 0: return x0;
            case 1: return x1;

```

```

        case 2: return x2;
        case 3: return x3;
        case 4: return x4;
        default: throw ArrayOutOfBound(i);
    }
}
};

//...DEFINITION OF THE NUMBER OF DIMENSIONS

//ORIGINAL SCHEMA...
class ANNpointObjPers : public ooObj {
public:
    int index;
    int plate;
    int fiberId;
    int run;
    int rerun;
    int camCol;
    int field;
    int id;
    int parent;
    int row;           // objc_rowc
    int col;           // objc_colc
    double ra;
    double dec;
    int objc_type;     // photo object type
    int type_g;        // photo object type
    int type_r;        // photo object type
    int type_i;        // photo object type
    char objtype[10];  // from plug map file
    int primTarget;    // target selection flags
    int secTarget;     // target selection flags
    float spectral_class;
    float z;
    float z_err;
    float z_confidence;
    int z_status;

```

```

float r_petro;           // r' petrosian magnitude, dereddened
float r_psf;            // r' psf magnitude, dereddened
float r_dev;           // r' dev magnitude, dereddened
float i;               // i' petrosian magnitude, dereddened
float i_psf;          // i' psf magnitude, dereddened
float i_dev;         // i' dev magnitude, dereddened
float ug;
float gr;             // model colors, dereddened
float ri;
float iz;
float ug_err;
float gr_err;
float ri_err;
float iz_err;
float psf_ug;
float psf_gr;        // psf colors, dereddened
float psf_ri;
float psf_iz;
float psf_ug_err;
float psf_gr_err;
float psf_ri_err;
float psf_iz_err;
float petrorad;      // r' petrosian radius
float petror90;     // r' 90% light radius
float petror50;     // r' 50% light radius
float ir_dev;       // i' deV radius
//...ORIGINAL SCHEMA

//DEFINITION OF THE SPACE...
#define FACTORS_TYPES_NUM 3
enum scaleFactors {DEGMPC=1, GR=2, RI=3};

ANNcoord fetch(int i) {
    switch (i) {
        case 0: return (ANNcoord) (z/3e-3);
        case 1: return (ANNcoord) (ra*scale(DEGMPC,z));
        case 2: return (ANNcoord) (dec*scale(DEGMPC,z));
    }
}

```

```

    case 3: return (ANNcoord)((gr-scale(GR,z))/0.1);
    case 4: return (ANNcoord)((ri-scale(RI,z))/0.1);
    default: throw ArrayOutOfBounds(i);
}
}

private:
double scale(int fac, float z) { // simple lookup table
    static double lookup[][FACTORS_TYPES_NUM + 1] = {
        // z      DEGMPC      GR      RI
        {0.01,1.92859542953 ,0.738,0.449},
        {0.02,0.97596732009 ,0.761,0.441},
        {0.03,0.658468820282 ,0.781,0.436},
    [...here the function is cut...]
    return (z == max_z) ? lookup[idx_z][fac] :
        (lookup[idx_z+1][fac]-lookup[idx_z][fac]) *
        (z-lookup[idx_z][0])/step_z +
lookup[idx_z][fac];
    }

//...DEFINITION OF THE SPACE
};

```

References

ⁱ <http://projects.fnal.gov/act/kdi.html>

ⁱⁱ www.labs.redhat.com/orbit/ or www.ooc.com