LA-UR- 93 - 587

CONF-930206-9-4

Title: ADLIB - A SIMPLE DATABASE FRAMEWORK FOR BEAMLINE CODES

LA-UR--93-587

DE93 008693

Author(s): C. Thomas Mottershead

Submitted to: Computational Accelerator Physics Conference
Hilton at the club
Pleasanton, CA
February 22-26, 1993

RECEIVED
FEB 04 1993
OSTI

MASTER

Los Alamos
NATIONAL LABORATORY

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by the University of California for the U.S. Department of Energy under contract W-7408-ENG-36. By acceptance of this article, the publisher recognizes that the U.S. Government retains a nonexclusive, royalty free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. The Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy.

ADLIB - A Simple Database Framework for Beamline Codes*

C. Thomas Mottershead

AT-3, MS H808, Los Alamos National Laboratory,
Los Alamos, NM 87545
(505) 667-9730
email: motters@atdiv.lanl.gov

Summary

There are many well developed codes available for beamline design and analysis. A significant fraction of each of these codes is devoted to processing its own unique input language for describing the problem. None of these large, complex, and powerful codes does everything. Adding a new bit of specialized physics can be a difficult task whose successful completion makes the code even larger and more complex. This paper describes an attempt to move in the opposite direction, toward a family of small, simple, single purpose physics and utility modules, linked by an open, portable, public domain database framework. These small specialized physics codes begin with the beamline parameters already loaded in the database, and accessible via the handful of subroutines that constitute ADLIB. Such codes are easier to write, and inherently organized in a manner suitable for incorporation in model based control system algorithms. Examples include programs for analyzing beamline misalignment sensitivities, for simulating and fitting beam steering data, and for translating among MARYLIE, TRANSPORT, and TRACE3D formats.

The Problem With Big Codes

At present, many independent and well developed beam optics codes exist. They are typically large, self-contained, batch-oriented programs that communicate via their own uniquely formatted input and output files. The input languages vary from simple and rather limited to very complicated and powerful. The input parser is the code that reads this input file, and stores the commands and machine parameters in some kind of data structure, usually FORTRAN common blocks, where they can be accessed by the computational routines. Depending on its power and complexity, the input parser itself can be a substantial piece of code, sometimes a significant fraction of the whole code. People tend to use the code(s) they know best because of the substantial investment of time required to develop the expertise necessary to use them effectively. But none of these codes have all the desired capabilities; some do space charge to first order, some do high order without space charge, some track particles, while others deal with misalignments. The complete design process often requires all of these capabilities and more. Error studies can be particularly tedious, requiring many runs of the code, interleaved with minor editing of the input file and tabulation of results from the output file. The process cries out for automation, but modification of these big codes is difficult, often even for the author. There is always the risk of damaging existing features in the effort to introduce new ones. The desired new functionality may even be impossible, if it clashes too much with the internal structure of the

* Work supported and funded by the US Department of Defense, Army Strategic Defense Command, under the auspices of the US Department of Energy

code. While some codes, such as MARYLIE, allow private USER routines, kept in a separate source file, as a mechanism for adding functions compatible with the basic structure of the code, in most cases modifications would result in a private version of the code with undocumented features that are wiped out by the next official distribution of the code from its keeper. If still needed, the modifications would have to be repeated for the new version of the code.

The Virtues of Small Codes

When we need to write a small specialized physics code to do a well defined job in the simplest way possible, we would like to begin with the machine parameters already available in a known data structure, and to reuse well proven subroutines for beamline elements, mathematical operations, and the like, rather than develop them anew from scratch. This is, of course, the motivation for modifying an operational code, despite the hazards mentioned above. But what we need is the kind of modularization that would permit us to concentrate on the problem solving, without having to worry about the overhead of modifying a large code containing many complex functions extraneous to the job at hand. This paper describes an attempt to move in this direction, toward a family of small, simple, single purpose physics and utility modules, linked by an open, portable, public domain database framework. While such codes are easier to write, their big virtue is that they are so small they may be understood whole, and be copied and modified to create a new small code that serves another special simple purpose, but shares the same data structures, and perhaps many subroutines. But there is no risk of damaging the parent code, which remains available in its debugged state as a check on the data, as well as for its original purpose. As the inventory of these little codes grows, the whole system becomes more powerful and flexible, but in bite-sized pieces. Examples from the current inventory are discussed below.

Modularization

This modularization of the physics codes, while valuable in its own right, also produces code that is inherently organized in a manner suitable for incorporation in a scheme of the sort illustrated in Figure 1., which shows how optics codes could contribute to, and benefit from, a larger design, simulation, and operational control system for a real machine. The key is to develop a formulation of the machine and beam model databases so that the physics part of the optics codes can be separated from both their inputs and their outputs. The optics modules, which may in general be thought of as transforming the initial beam model B_i into the final beam model B_f , draw their parameters from, and store their results in, the database. Other modules put parameters into, and display results out of, the database.

Modern User Interfaces

One way to put parameters into the database is through one of the elegant, powerful, and user friendly interfaces being written these days on modern workstations for use as control system consoles. These use multiple windows, icons, pointing devices, colors, and interactive graphics to display and reset machine parameters via a set of database access routines. The control system in turn, runs the actual hardware from the contents of the data base, shown in the diagram as the link to the "real machine". Some

Optimizer Modules:

The second way to change the machine parameters is via the optimizer module, which should contain callable subroutines that take in selected merit functions, and deduce from them recommended new values for selected machine parameters in an effort to drive the merit functions toward zero. The new machine parameters are then loaded into the database for use in computing the next iteration of the merit function. Most available optimizers and equation solvers are written to be in control, calling the function(s) to be solved or minimized as often as they please, in an effort to maximize their internal efficiency. For our purposes, this is backwards: a good optimizer or fitter should be callable, leaving the external program in control, and extremely frugal with function calls, because these can each be a very expensive model computation, or even an experiment. We have written three such routines: AMDII is an efficient non-linear system solver using a new adaptive multidimensional inverse interpolation algorithm; NLS is a similar non-linear least squares routine; and QSO is a new quadratic step optimizer that uses quadratic fitting of the Hessian to the best previous points to speed convergence.

Translators:

The third way to load machine parameters into the database is from an external file such as the input deck for your favorite major code, shown in the figure as "code A". The routine that reads this file is the input parser from code A, which could be removed surgically and used to initialize the database. This separation of the input parser may well be a non-trivial exercise. In the case of MARYLIE, it has so far proven easier to write a special USER routine to dump the current beamline to a simplified ASCII file. A short stand alone program (DMPIN) was then written to read this dump file into the database. Another program must then be written to generate code A's input deck from the current contents of the database. As other operational codes receive this treatment, the set of programs to map their various input file formats to and from the database will, in itself, constitute a valuable translation facility. At this point, code for translation between TRACE3D, MARYLIE, TRANSPORT, and TeX tables is partially operational.

Internal Data Structure

The database itself is the backbone of this entire development. Its implementation must be simple, practical, portable, flexible, and efficient, with well-defined units and variables, capable of describing a general beamline and a general beam. Portability means standard FORTRAN 77 common blocks, without computer specific extensions. Generality means to be inclusive enough to support all the commonly used formulations of accelerator physics. That is, while the physics modules themselves may use very different techniques, the machine and beam parameters they need should all be present in the common data structure. Of course it is difficult to anticipate every future need. So the formulation reported here is to be regarded as experimental, and is expected to evolve in the course of working on actual problems, hopefully toward the simplest, most practical, convenient, and generally useful definitions.

The ADLIB Library

A set of database service routines is needed to make data access flexible and portable. Portability is enhanced by isolating the implementation details of the database from the application codes that use it. ADLIB is a prototype open database library consisting of a small number of FORTRAN routines to store and fetch the simplest possible data structures. The routine names (almost) all begin with the prefix AD, for Accelerator Database, to identify and distinguish them. The whole library is called ADLIB, which also means: to improvise. Internal storage is in standard FORTRAN 77 common blocks. External storage is in a direct access binary file, for fast I/O. The data structure consists of an 8 character name, a block of integers, and a block of double precision real parameters. The block of integers is meant to hold various type codes, flags, and indices. So far only the first four integers are defined in general:

1. NI = Number of integers stored in data item. NI is at least 4, but may be as large as necessary.
2. NR = Number of real parameters stored in data item. Unrestricted.
3. MAJOR = Major type code (e.g. magnetic element).
4. MINOR = Minor type code (e.g. subtype in MAJOR).

Beyond this, ADLIB itself doesn't care what the parameters mean, and in other languages and systems, the data structure could be handled differently. The routines defined so far are:

ADINIT: This subroutine initializes (clears) the internal common blocks to zero or blank, for fresh starts.

ADREAD(lun,fname,info): This subroutine uses logical unit **lun** to load the internal commons from the direct access binary disk file named **fname**. The integer **info** will be an error message. Any code that begins with a call to **adread** will have access to the parameters stored in the specified copy of the database.

ADSAVE(lun,fname,info): This subroutine reverses the process, using **lun** to write the contents of the internal commons back to binary file **fname**. Any code that ends with a call to **adsave** will store its results in the specified copy of the database.

ADNEW(ni,nr,lib,kxb,nuslot): This subroutine creates a new storage record in the database containing room for **ni** integers and **nr** real parameters. The slot number assigned to the new item is **nuslot**. This is normally the next available location in the database, but if the database is full, it is returned negative as an error message. The integer block offset **lib**, and real block offset **kxb** are returned only for the convenience of users who want to manipulate the commons directly, without the overhead of a subroutine call. This is an example of the open, permissive, philosophy that aims to put the user in complete control of his own code.

ADPUT(nslot,cname,iblok,rblok,info): This subroutine stores the complete record consisting of the character*8 name **cname**, the block of integers (**iblok(j),j=1,ni**), and the block of double precision reals (**rblok(j),j=1,nr**) into database slot **nslot**. The error flag **info** is returned to indicate the success of the operation.

ADGET(nslot,cname,iblok,rblok,info): This subroutine is the inverse of ADPUT. It retrieves the complete record consisting of **cname**, the **ni** integers in **iblok**, and the **nr** double precision reals in **rblok** from database slot **nslot**.

ADPOKE(nslot,nvar,rval,info): This subroutine stores a single real value **rval** into location **nvar** of database slot **nslot**.

ADVAL(nslot,nvar): This double precision function returns the **nvar**-th real parameter stored in database slot **nslot**. It is the inverse of **ADPOKE**.

NUMAD(nslot,nvar): This integer function returns the **nvar**-th integer stored in database slot **nslot**. It is useful for rapid scans of the database for a particular integer flag, but has no inverse because the stored integers are generally type codes and dimensions that should not be easy to change.

Type Codes

As in all optics codes, the meaning of the parameters is specified by type code flags, and is of course crucial to their use in the physics programs. Here we have allowed both a **major** type code in **iblok(3)**, and a **minor** type code in **iblok(4)**. But since the application programs are small and specialized, they do not necessarily need to cope with the general case. A program that does not use a particular data type can be written to simply ignore it. Control of this rests with the particular program, not the database library. New type codes may therefore be defined as needed, hopefully in an orderly and backward compatible manner. All type codes have not yet been fully defined, because the need has not yet arisen. The tentative definitions being tried at the moment are:

Drifts: MAJOR=1,
MINOR=0 for a normal drift,
=1 for drift to a special checkpoint, (e.g. a diagnostic device).
NI=4, The minimum number of integers possible.
NR=2, (aperture and length in meters)

Linear Magnetic Multipoles: MAJOR=2,
MINOR=Maximum multipole component present
(quadrupole=4, sextupole=6, octupole=8, etc.)
NI = at least 4, plus fringe field type flags
NR = MINOR+6 to allow for misalignment parameters
(Dx, Dy, Dz, Roll, Pitch, Yaw), plus aperture,
length, and normal and skew components of the
field strength for each multipole present.

Magnetic Bends: MAJOR=3

This type code is not yet fully defined, but the parameter set should be inclusive enough to allow the physical description of all the bend magnet types in common use.

Beams: The three major possibilities for representing an accelerator beam are particle sets, moments, and distribution functions. The major type code could flag these three major representations, while the minor type code could indicate the storage mode and format, either in external files, or in the database itself if it would fit. For external files, the name field could be used to generate the file name in a systematic way.

Particle sets, represented by arrays of canonical MKS particle coordinates, could be stored in MARYLIE type ASCII files, or PATH type binary files. If the set were stored in the database itself, the coordinates would go in the real block, and we would have NR = six times the number of particles.

Moments: The usual beam sigma matrices are the second moments of the beam distribution. Codes such as BEDLAM extend this concept to higher orders. Giorgelli indexing for monomials is used in both MARYLIE and BEDLAM to store Lie polynomial map coefficients and higher order beam moments. This same sequencing is natural for the database as well.

Phase Space Distribution Functions: There are few practical analytic formulations for realistic phase space densities. Gaussians are often used for their simplicity, and are completely described by their zeroth, first, and second moments. Other analytic formulations could be parameterized similarly.

ADLIB Utility Programs:

The following programs are available for manipulating the database itself, which is normally stored as a binary file for efficient I/O.

ADOUT writes the database to an ASCII file suitable for human reading and modification with a text editor.

ADIN loads the above human readable ASCII file into a copy of the database.

ADUMP writes the database to a concise and complete ASCII file which, if not particularly human readable, is at least suitable for long term storage and email.

AEDIT is still under development, but will allow direct editing of the binary database, including adding and deleting beamline elements, and defining indirect parameter linkages representing constraints.

ADLIB Physics Applications:

The database contains a general machine description consisting of element sequence arrays, stored with a simple, flexible, and efficient indexing scheme for accessing element parameter blocks. While the indexing scheme may be used directly by the application codes, it has been packaged for convenience in the database access routines that constitute the ADLIB library. These routines are all written in FORTRAN77 on a VAX, and should be used to develop database oriented physics codes. Then one could expect that only the ADLIB routines would need to be rewritten to implement this database in other languages and operating systems.

At this point, the storage of beam parameters and control structures in the database is quite rudimentary. This is expected to evolve with usage. In fact the only control structure implied by the database is to consider the beamline to be defined by the list of elements in the order of their storage. The ability to describe repeated (nested) groups of elements in the database

has not yet been defined. The application codes can, of course, do anything they please with this data. So far, a simple do loop through the entire beamline has sufficed for most applications. About 20 ADLIB application programs have been written to date, mostly by modifying a predecessor program, sometimes extensively. The following are some representative examples:

DMPIN loads a MARYLIE beamline into the database using a simple ASCII dump file written by a MARYLIE user routine.

MATRIX interactively displays the R-matrix between any two selected points of the beamline in the database.

ADVIBS does a vibration sensitivity analysis of the entire beamline by computing the final beam steering angle induced by misalignment of each quad in the line. The sensitivity coefficients are written in a TeX table that may be formatted and printed using TeX, without further editing.

SIMS simulates beam steering due to quadupole rolls and displacements, and reports expected beam position monitor readouts at all indicated locations in the beamline. All quadrupole gradients and misalignments, along with input beam centroids, can be interactively changed.

STEX writes a ready to print TeX table containing steering sensitivity coefficients, at all the diagnostics, to movements of two steering quadrupoles in a particular beamline.

GENFIT uses the non-linear system solver AMDII to adjust selected database parameters to fit measure steering data for the above system.

The point is not so much what particular little job these small codes do. It is that they are small, were written relatively quickly, and could be readily cloned and modified to do something else. Thus the capability of the entire family is expected to grow, filling in the gaps between the major design codes, and perhaps eventually becoming a serious contender themselves.

Conclusion

None of the major optics codes provides all of the capabilities needed for complete design and simulation. Consequently, there is often the need to translate beamline descriptions between codes, as well as to write small specialized adlib codes to fill in a missing functionality. In addition, there is both an opportunity and a need to work toward a broader synthesis of accelerator design, simulation, and control systems, with prospective benefits on all sides. Full implementation of this grand design is a large undertaking, but plans can be laid so that small practical steps accumulate in that direction. This paper reports one of the most basic of these steps, namely an experimental prototype of the machine data base.

References

MARYLIE
BEDLAM
Georgelli

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the