NLCDB System Requirements

NLC Database (NLCDB) System Requirements

Initial Public Release 10-Sep-1999

1 Introduction

The design and construction of the Next Linear Collider is a very large physics and engineering project with many technological challenges. During the course of design, construction and operation there will be enormous amounts of data and documentation that must be archived, categorized, sorted, graphed, modified and most of all, easily retrieved and examined.

The NLC is a cooperative development project between Japan's KEK laboratory and SLAC (and other labs (?list here?). As such, there will be a need to easily communicate and share data across the Pacific and most likely, around the world. Of particular importance is the easy accessibility of design and engineering data across the Pacific.

Because of its size, the NLC itself will likely be constructed at a site perhaps far removed from the present location of either laboratory. This also means that both engineering and operational data must be easily available to Physicists and Engineers who may not be physically present at the NLC site.

1.1 Terminology

In typical fashion, this document contains a list of requirements that the NLCDB must or may ultimately meet. These requirements come in several industrial strengths:

1.1.1 SHALL.

This is a requirement that must eventually appear in the 'completed' system. Without it, the system functionality would be seriously diminished or made unusable.

1.1.2 SHOULD.

This requirement is highly desirable but may be completed at a later time. It could be dropped given compelling reasons without seriously impacting the system's overall functionality.

1.1.3 MAY.

This is a feature that seems useful. It may be included if time and resources permit. I suppose this means that these features will likely not get done unless

their importance gets elevated. This category can be used for those features which we might like to implement at a later time.

1.2 Existing systems at SLAC and elsewhere

For the PEPII project at SLAC, an attempt was made to integrate the viewing of much of the data starting from a single Web page located at

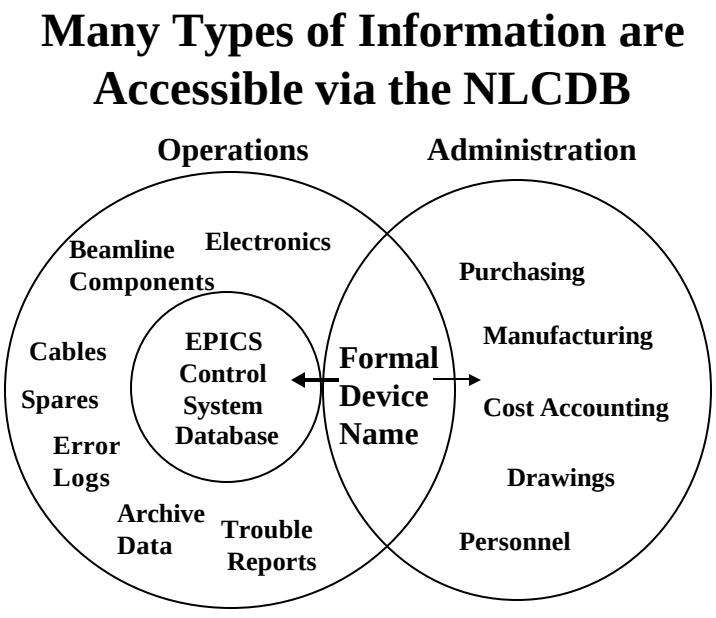
http://www.slac.stanford.edu/accel/pepii/db.htm.

The need for this was driven partially by the project personnel themselves who came to Andrea Chan in SCS asking for help in organizing their data. The result was partially successful, and we hope to learn from their efforts.

Other accelerator facilities seem to have been fairly successful storing the relevant data in an Oracle database and presenting it via a Web interface. We have been, and will continue to, look at their systems and communicate with the relevant people. These include but are not limited to BESSY, CERN & Fermilab.

1.3 NLCDB Information Overview

The information accessible via the NLCDB encompasses a wide variety of technical and administrative data. The following Figure should give you the big picture.



The information spans many organizations and disciplines. How the heck are we gonna do all this?

2 General System Requirements

2.1 User Interfaces

The NLCDB system is intended to be the central mechanism for reading and updating data relevant to the design, construction and operation of the NLC. These data must be readily available to all legitimate members of the High Energy Physics community involved with the NLC. In addition, the system structure and user interface must allow easy linkage across different types of data.

Examples would be to link components and their engineering drawings; publications and people etc. All of the above considerations lead us to the following requirement:

2.1.1 User interaction with the NLCDB **shall** be primarily via a web browser interface.

The Web seems like the only mechanism currently available to meet data accessibility requirements and allow access across all of the platforms common in the physics community. It's clear that the tools and technologies available for interacting with data via the Web will continue to develop for the foreseeable future.

While the Web will be the primary user interface, we may want forms or other mechanisms for database update in special circumstances.

2.1.2 The NLCDB **shall** provide the ability to import data from external sources.

Examples of these include test instrument output, bar code readers or files produced by other software packages. Software packages would include standard ones like spreadsheets as well as special purpose packages like Matlab or individual user written programs. In principle we need the ability to import data from almost anywhere, regardless of its source.

2.1.3 The NLCDB **shall** provide read and update access to parts of the database even when network connectivity is unavailable.

The NLC will be approximately 35 Km long. There will be many parts of the tunnel, alcoves and other areas where network connectivity may not be easily available. Because so much information is contained in the database that must be accessible by maintenance and other personnel, there must be a mechanism for users to have local access to a copy

of relevant parts of the database for reference and updates. These updates can then be loaded into the central database when network connectivity is again available.

2.1.4 Where possible and appropriate, a pictorial or graphical interface **should** be available.

The user **shall** be able to traverse the NLC by clicking on drawings or photographs, zooming in, selecting device lists, etc.

2.2 Database

2.2.1 Data in the NLCDB **shall** be stored primarily in the Oracle RDBMS.

Stanford and SLAC have a site license agreement. We have considerable in-house Oracle expertise available, as it is being used by other organizations within SLAC. Other HEP laboratories like CERN and BESSY also use Oracle. Second tier RDMS companies like Informix and Sybase have fallen on hard times leaving Oracle in a dominant position.

It may be reasonable later on to consider a pure Object-Oriented database like Objectivity for highly structured, large volume data like error logs or history buffers. The next version of Oracle (version 8) will incorporate object oriented features that may come in handy. Time and BaBar will tell if this is a reasonable path to follow or abandon.

2.3 Security

Since the recent break-in to the SLAC computing system, security has been a large concern in many people's minds. User validation initially means an encrypted userid/password that should be adequate for our needs. It seems that most technical data can be made available in a read-only mode unless it specifically contains information that might compromise SLAC security.

Regardless, we need to be flexible in constraining access to the database.

2.3.1 The NLCDB **shall** allow read access within or outside of the SLAC domain.

We need to have control over what data are generally visible only to the SLAC community and what can be visible to The Universe at Large a la Virginia Trimble.

2.3.2 The NLCDB **shall** allow write access within or outside of the SLAC domain.

Similarly, owing to the international nature of the project, some, but not all, data must be writeable from outside of the SLAC domain.

2.3.3 Write access to the NLCDB domain **shall** require encrypted user validation.

The encryption method shall be available in all countries from which institutions are contributing to the NLC. The encryption shall be adequate for security needs without requiring large computing resources. The encryption method will change if there are increased security issues, or as improved technologies become available.

2.3.4 Data shall be owned by specified users, according to data type.

This protects data from unintentional modification and identifies who shall update certain data. The method to control ownership shall be flexible.

2.4 Versioning of NLCDB data

This is a requirement that has come up repeatedly in many discussions. The consistent analogy is a code management system where the source code is stored in such a way that by default you always get the latest version, but previous versions can be recovered, differences between versions can be examined, obsolete versions can be purged, and there is a history of who made each change. In addition, groups of code modules of a specific version can be saved as a named unit.

2.4.1 The NLCDB **shall** save a time-stamped history of changed data, and who changed the data.

This provides basic version control that can be identified by time. The history of who changed the data is useful to debug the database and accelerator systems that depend on it.

2.4.2 The NLCDB **should** allow the identification and naming of functionally related data elements, and the ability to save and restore versions of these named groups.

In addition to a timestamp, there may be a need to identify sets of records by specific group and version identification. An example might be all the data in a specific modeling deck. The NLCDB **should** provide for named collections of data with a given timestamp or version ID.

2.5 Failure Recovery

2.5.1 The NLCDB **shall** never lose more than the last transaction in the event of a hardware or software failure, or a natural disaster.

With full and incremental backups, data mirroring plus transaction journals, this is just de rigueur with modern database systems and we should do no less.

2.5.2 Data backups **shall** be stored in a remote secure facility for recovery in the event of a large-scale system failure.

This is fairly standard practice and since so much is riding on this database, we need to ensure that this is done.

2.6 System Availability

Due to the potential international nature of the NLC project, the NLCDB will need to be up and running during working hours across many time zones. We will need to design our system maintenance and backup activities to minimize system downtime for as many users as possible.

2.6.1 The NLCDB **should** be available on a 24 X 7 basis except for scheduled outages.

This may be hard to achieve but we should try to come close.

2.6.2 NLCDB system outages and upgrades **shall** be coordinated with all laboratories involved.

Of course there will need to be scheduled system upgrades and maintenance. We'll need to work out arrangements with our system and database administrators to minimize the impact of these outages.

3 Database Content and Links

This section focuses on each type of data that will comprise the NLCDB. Other groups both inside and outside of SLAC will maintain many types of data. Regardless, all of this data shall be easily accessible from the NLCDB Web pages. A general list of the types of data to be included in, or linked from, the NLCDB would include:

- Component and device information including history & measured parameters.
- Travelers (eg. for magnets)
- Cabling & routing information.
- People data such as project personnel, committees, contact information etc.

- Engineering drawings & specifications.
- Publications associated with the NLC and its development and construction.
- Static & dynamic Control System data.
- Data and scripts necessary to generate & maintain parts of the Control System Database.
- Cost accounting & financial data
- Trouble Reports and maintenance statistics.
- Networking and computer equipment information
- Spare parts & Inventory.

As mentioned above, since NLC physics and engineering development is already well underway, we will start with development of component data, adding others as we go.

3.1 Device Data

Device data includes data associated with the logical and physical components of the NLC. Sample device categories include:

- Beamline optics
- RF systems
- Vacuum systems
- Power supplies
- Computer crates and modules
- Cables
- Feedback loops
- EPICS database generation
- Calculations

On the most primitive level, device data consists of:

- Information that defines the identity, classification and function of each device.
- Static parameters and descriptive information.
- The location of the device within the NLC (or spares).
- Measurements of functional parameters.
- Costing and manufacturing data.
- Relevant Control System parameters.

What do we do about logical devices for control and operation i.e. feedback etc.? Should we have a separate section? Same formal device name format?

3.1.1 All device data **shall** be stored in, or accessible from, the NLCDB.

3.1.2 Each logical device **shall** have a formal device name that is unique within the NLCDB.

This is our much-discussed Formal Device Name.

3.1.3 A unique system-generated ID (UID) **shall** internally identify each logical device.

This allows tracking of device information even if the Formal Device Name changes due to a change in physical position.

3.1.4 The physical device associated with each logical device **shall** have a separate identifier.

It is important to note that the formal device name identifies a logical device. The physical device that resides there may change. Some devices like magnets will almost never be changed out once they are installed or don't have spares at all. Other devices will have numerous spares and will be replaced as needed.

3.1.5 Device hierarchy and connectivity **shall** be stored in the NLCDB.

Devices are atomic entities but they are linked physically and/or functionally to other devices. Since we cannot have reasonably short device names that describe many levels of hierarchy and connectivity, this must be information that is stored separately in the database.

3.1.6 The NLCDB **shall** contain, or link to, the types of data found in these existing standalone SLC database systems:

- CATER (problem reporting),
- CAPTAR (cabling),
- DEPOT (crates and modules and their maintenance),
- CANDO (network and node information).

3.1.7 The NLCDB **should** contain the machine parameter list.

3.1.8 Database entry has failsafe choices.

There shall be failsafe choices and checks, so that a user cannot take shortcuts in data entry, giving inconsistent or different definitions for a device. Anyone who enters a given type of device should achieve the same device structure.

3.2 Links to External Database Systems

In addition to data that will reside directly in the set of tables that we call the NLCDB, there is a large amount of additional data, collected and maintained by other organizations both in and outside of SLAC. This data is related to that in the NLCDB proper and so we must have access to it.

3.2.1 The NLCDB **shall** have links to all relevant mechanical drawings.

Engineering drawings are generated at multiple sites, and should be accessible from the NLCDB regardless of site of origin. SLAC is currently working on the purchase and implementation of an Oracle-based Project Data Management system (PDM) that will handle mechanical drawings and associated data.

3.2.2 The NLCDB **should** have links to all relevant electrical drawings.

This may be much more difficult since many different drawing packages are involved.

3.2.3 The NLCDB **should** have links to real-time data from the NLC control system.

This seems like a really nice idea and should be doable. Anyone want to up the ante?

3.2.4 The NLCDB **should** have links to history data from the NLC Control System.

This also seems like something useful but perhaps we could do without it initially.

3.2.5 The NLCDB **shall** have links to people data and BF Mail data.

This is intended to provide at least the functionality of the corresponding section of the PEP II Project database. It includes data like:

- List of project personnel
- Committees and meetings
- Communication information
- E-mail functions

3.2.6 The NLCDB **shall** have links to all relevant publications.

The PEP II project database stores data associated with publications: author, abstract and location. A useful feature for the NLCDB to add would be viewing of the actual document on-line. We may look at purchasing a document control system, or at borrowing from CERN's successful efforts.

3.2.7 The NLCDB **shall** have links to all relevant formal NLC documentation.

We will need help from publications on how to manage this.

3.2.8 The NLCDB **shall** have links to financial data.

This will include at least:

- Purchase requisitions and their status.
- Purchase order information that can be loaded from other institutions as well as SLAC to allow querying of requisition data across institutions.
- Work Breakdown Structure (WBS).

4 Functional Requirements

Now that we have all of the data, we need to define data relationships and functions that allow us to view and maintain it. These functions include data entry and updating, as well as flexible query and reporting capabilities.

Allowing users to keep their data current without permitting total data anarchy is an issue that must be addressed for each type or category of data. Some data can be updated automatically, perhaps via email processing. Others can be done via data entry applications by the responsible individuals. Still other data may be central to the integrity of the database as a whole and must only be updated with care and consent.

4.1 Data Query and Maintenance Functions

For the NLCDB we need to provide at least the following set of system interfaces and functions, integrated together in a consistent manner. Some of these correspond to existing standalone SLC database applications. We will need to develop a separate functional requirements document for each of these categories.

4.1.1 Device Data

One aspect of the NLCDB is that it can be seen as device oriented. One should be able to select and count devices based on name criteria, zoom in or "drill down" to find all of the

information about a specific device. But even more general queries are required since devices can be logically related even though that relationship is not reflected in their device names.

4.1.1.1 The NLCDB **shall** provide functions to count and retrieve lists of devices based on a variety of search criteria.

Examples of search criteria are NLC area, device type, wildcard pattern matching on formal device name, etc.

4.1.1.2 The NLCDB **shall** provide functions to count and retrieve lists of devices based on named groups of device types.

This means that we can for instance associate all devices that are magnets. Since their device name (QUAD, BEND, SEXT etc.) gives no clue that they are all magnets, this association must be provided by the database.

4.1.1.3 The NLCDB **shall** provide functions to find and examine all data about a specific device, or set of devices.

This is the “drill-down” or “zoom-in” capability which, starting with a given device or selected list of devices, allows one to view everything about it from its current status to its manufacturing, testing and operational history. But this amount of data can be huge. Therefore the following functionality is also necessary.

4.1.1.3.1 The NLCDB **shall** allow the user to select which device data to display.

The combination of the above functions allow one to query and retrieve a data set like the length, aperture and tilt for all the quadrupoles in the big bend region.

4.1.1.4 The NLCDB **shall** provide mechanisms for creating, maintaining and viewing physical and logical hierarchy, connectivity and functional associations between components in the NLCDB.

Devices do not exist in isolation. Both the logical and physical connectivity of all devices must be represented in the database. This allows retrieval of information like a list of all the cables that are attached to a specific BPM. This functionality also allows the user to follow links of connected devices through the system.

4.1.1.5 The list of device oriented functions will grow!

4.1.2 Cable Data: The NLCDB **shall** provide functions for input and maintenance of cables, their assemblies and connections (CAPTAR functions).

We should review all of the existing CAPTAR functions and add/modify them as appropriate. This means feedback from the existing user community and adjusting the requirements for our completely integrated system/ We hope!.

4.1.3 Crates, Modules, Maintenance Data: The NLCDB **shall** provide functions for input and maintenance of data about crates, modules, and maintenance records (DEPOT functions).

The same comments from CAPTAR apply here.

4.1.4 Problem Reporting and Tracking: The NLCDB **shall** provide functions for problem reporting and tracking (CATER functions).

Again the same comments as CAPTAR. Additionally, we should examine Remedy in some detail to see what functions it offers that we might like to have.

4.1.5 Computer and networking equipment tracking. The NLCDB **shall** provide functions for recording and tracking networking and computer equipment (CANDO functions).

4.1.6 Travelers: The NLCDB **shall** provide functions for input and maintenance of traveler data, including scanned images of the traveler documents themselves.

4.2 Data Loading

In addition to data entry and updating via Web user interfaces, the NLCDB system will need to provide ways to load data into the database from diverse sources, and in diverse formats. For some types of data we may not want to have users directly changing large amounts of data. For other types we will want users to be able to maintain their own data. We may need to write additional scripts for loading data provided by other organizations.

4.2.1 Beamline devices **shall** be loaded from files generated by MAD or related modeling programs.

This is the core of the NLCDB. Much of the information fans out from these data. The beamline is what the NLC is all about. All the other stuff is there to support it. Loading the modelling decks will define the initial set of devices, device types, and static parameters in the system, and will allow users to count and list NLC devices.

4.2.2 There **shall** be a Web interface for loading spreadsheet data into the database.

It is envisioned that this will be a pre-defined set of Excel templates for specific types of data entry. This is what PEP-II did with some success. It requires a special load program for each spreadsheet template.

4.2.3 There **shall** be methods for loading data from sources such as instrument and program output.

4.2.4 Individuals responsible for data maintenance **shall** be kept in the database.

It is important that who is responsible for what data is always well defined. Keeping data current is 90% of the problem. Also, the audit trail system will track who did what and when.

4.3 Data Export

Once the data is in the database, we also need to be able to display it, and export it in a variety of formats for use by other applications.

It's unclear at this point how elaborate we want to get but at a minimum we need simple tab-delimited spreadsheet output. In fact, spreadsheets are so ubiquitous, we have provided a special section just for functions to interface with them.

4.3.1 Specialized output functions **shall** be available to export data for specific needs.

One example is the translation of drawing numbers from out internal scheme to another format when we go out to bid for construction. (See Clay Corvin for details.) Another example is exporting to a format compatible with a specific data analysis package. Since special code will be required for each of these, there is certainly a cost/benefit tradeoff here.

4.4 Spreadsheet Functions

Spreadsheets are so ubiquitous in any engineering environment that they deserve their own section in this document, as a special class of data export and import tools. To meet many unanticipated user needs as well as to provide access to database information via well-known tool, we must have a spreadsheet interface to the database that goes beyond ingress and egress of data via the spreadsheet. What kinds of capabilities might we envision? Let's try these on for size.

4.4.1 As the minimum functionality, the user **shall** be able to direct output to a spreadsheet from an NLCDB query.

4.4.2 The NLCDB **shall** provide standard spreadsheet templates with built-in queries and calculations.

It may be easier to construct some simple applications or calculations by making a spreadsheet that can do database queries and perform some local calculations on the results. The individual user as desired can vary the nature of these calculations.

4.4.3 The user **shall** be able to store altered calculations & queries into a private space in the NLCDB.

The idea is to let users build their applications. Those that are generally useful can be migrated to a public place after screening. There's no way we can keep up with all of user's needs for data extraction and manipulation. Probably we need a mechanism for user's to submit clever things they've done so they can be made universally available.

4.4.4 Direct database access **shall** be provided for spreadsheets.

With ODBC installed on the user's workstation and VBA (Visual Basic for Applications), direct database access is possible. With the many plotting and graphing functions available from Excel, this may be a very useful way to build some custom applications.

4.5 Reporting

There are many third-party reporting tools that can be employed to extract data from the NLCDB. It's not clear at this time how much functionality we must directly provide and how much can be done by users with Oracle or third-party packages.

This extends to the construction of standard pie charts or bar graphs etc., all of which can be produced if data can be exported to a spreadsheet.

Perhaps we'll need to have a canned set of reports but what these might be are unknown at the present time.

4.6 Audit Trails

We need functions to data change histories. BESSY has a nice mechanism for this so here it is.

4.6.1 The NLCDB **shall** provide functions to track changes to the database.

For BESSY you can track INSERT, DELETE, UPDATE, EXECUTE, CREATE, DROP & GRANT operations by user, timestamp and object (table or view) name.
We should do no less.

4.7 EPICS Database Generation

We want to have all of the EPICS database information generated from the NLCDB. It is the "master" from which all control system data are generated. Additionally, all generic displays e.g. all XCOR's in a sector, and other EPICS data files will be generated automatically. This is the only way we can guarantee consistency in this very large EPICS database.

4.7.1 All non-runtime EPICS data **shall** be stored in the NLCDB.

This includes all data necessary to generate the downloaded .db files, monitor limits, save-restore templates and generic displays.

4.7.2 Quasi-static data like alarm & monitor limits that are required to generate the EPICS database **shall** be updated from EPICS to the NLCDB.

The idea here is to keep all data necessary for the generation of the EPICS database current. This includes operator changeable limits etc. This of course does not include real-time data.

4.7.3 NLCDB tools should automatically generate basic EPICS data files for generic displays, alarm, archiving and other required data files.

We need to automatically create EPICS data files for generic facilities like displays according to device type. For example, a generic display of all magnets would automatically include new magnets and remove deleted magnets. The archive facility must automatically include/exclude devices as they are defined and deleted. So any control files used by the archiver must be automatically generated. Other EPICS facilities will also need automatically-generated files.

This will be a critical tool to allow scaling EPICS to a large system like the NLC. While manually created displays and other EPICS files will be needed, we must avoid the time and labor expended to manually create and maintain all files.

4.7.4 Conditions and states that must be constant across IOC reboots **should** be saved in the NLCDB.

These data are presently held in save-restore files. We might leave them there. Or not.

4.7.5 Validation of EPICS database

Generation of the EPICS database from Oracle shall include data checking and validation so that we get a usable error-free version of the EPICS database.

4.8 Error Logs

Do we want to commit to this? If so we need some requirements.

4.9 History Buffers (Archiver)

Is it worth trying to use Oracle to store the archive data? What are the advantages and disadvantages? Add some requirements here or delete this section.

4.10 Miscellaneous Functions

4.10.1 The NLCDB **may** provide functions to generate modeling input decks.

4.10.2 The NLCDB **may** provide computational functions to calculate twiss parameters and other modeling data.

4.11 System Integration

The PEPII relational database project had many successes, but it has also taught us that the database data can easily become obsolete and confusing.

NLC project will need system integrators that confirm that systems are defined correctly and work correctly. The NLC database contains a lot of data, but it can easily become meaningless unless someone confirms that it is up-to-date: that the correct devices are defined, that the device parameters are correct, and that the database tracks all changes. For example, an integrator must also confirm that the NLC device subsystems, which use the data, work correctly; and someone must confirm that old device (units) are deleted. Assigning owners of device subsystems will help solve correctness of data, but the integrator will confirm that the whole subsystem works.

This document does not attempt to assign this task. The NLC project administration must identify which person or people are responsible. It could be the area manager or subsystem integrators specific to the subsystem; and possibly with help from database group. We only state that someone must act as integrator so that we continue to have correct data in the database.

Identify the system integration role early. If the role doesn't exist, the software group will become the de-facto integration team. When people are commissioning or controlling devices, the interface they see is the software and the database. If anything goes wrong, they often assume that the software/database that is the end tool controlling the device is at fault, so they contact the software group. But if the owner of the device entered garbage in the database, or if there is a hardware problem, etc., then the subsystem will not work.

We also will need good commissioning tools. Note that in the SLC world, CAMCOM was sometimes misunderstood, and hence not used by commissioning teams. We need good, understandable testing tools. Otherwise, people will turn to the database group to interpret database values and debug device systems. We must give them useful tools so that they can solve many of the problems on their own.